TRANSACTION DATA STRUCTURE
FOR PROCESS COMMUNICATIONS AMONG
NETWORK-DISTRIBUTED APPLICATIONS

CROSS-REFERENCES TO OTHER APPLICATIONS

The subject matter disclosed in this application is related to subject matter disclosed in concurrently filed, commonly-assigned U.S. Patent Application No. 08/XXX,XXX entitled "Interaction Protocol For Managing Cross Company Processes Among Network-Distributed Applications" (hereafter, "the Protocol patent application"), and to U.S. Patent Application No. 08/YYY,YYY entitled "Shared Transaction Processing in a Clustered Process Automation Environment". These disclosures are incorporated by reference herein for all that each teaches as if set out in full.

FIELD OF THE INVENTION

The present invention relates generally to systems that manage transaction processing message flow in a distributed computer network such as the Internet. In particular, this invention provides a transaction processing system and method that make use of a novel transaction data structure for implementing transaction processing between network-distributed software applications.

BACKGROUND OF THE INVENTION

Business entities have long recognized that substantial productivity and marketing benefits may potentially arise from conducting commercial business activities and

1 business processes over distributed computer networks. In order for a business to achieve

2 the full benefits of network-based commercial activity, the firm's existing commerce-

3 related or business process software application systems must communicate both among

4 each other and with the application systems of other business entities. Earlier efforts at

5 business-to-business commerce activity, such as those led by Electronic Data Interchange

6 (EDI) applications for example, focussed on high volume transaction processing for large

7 firms. Because of incompatible application file formats and communications protocols,

8 and requirements for expensive application programming changes to existing systems,

9 EDI applications were largely viewed as being commercially practical for only the largest

10 companies and for only a select number of applications. Moreover, because of a lack of

11 any universal data interchange formats, companies were, and still are, often prevented

12 from exploiting their own enterprise systems integration to reach external partner

13 applications. As a result, a business may need to spend substantial time to extract,

14 redefine, and update data to serve specific collaborative needs with partners or customers.

15 In addition, smaller companies with limited information technology development budgets

16 or with old legacy systems may still be struggling with internal business systems

17 integration issues.

18   In recent years, the Internet distributed computer network has developed the

19 infrastructure and data communications protocols to connect all businesses to each other

20 regardless of their size, geographic location or position in the supply chain. The Internet

21 is a collection of interconnected individual networks operated by government, industry,

22 academia, and private parties that use a set of standard data communications protocols to

1   form a global, distributed network. Networked distributed computer systems may be

2   configured as intranets, extranets or publicly available systems using Internet

3   technologies. Internet technologies provide business entities with another opportunity to

4   achieve substantial productivity gains and marketing benefits by conducting internal,

5   business-to-consumer and business-to-business Internet-based commercial activities

6   among employees, and with customers, vendors, suppliers and other parties related to

7   their business enterprises. Internet-based commercial activities, referred to generally in

8   the current literature as *"electronic commerce"*, *"e-commerce"*, or *"e-business"* include,

9   but are not limited to, all types of business processes that can take place in a secure

10   manner online, as well as the more traditional buying and selling of goods and services.

11   The Internet environment holds out the promise of true collaborative data exchange and

12   software application interoperability for business firms of all sizes.

13   Several standardization efforts by industry consortia and e-commerce vendors are

14   underway in an effort to achieve Internet application interoperability and seamless

15   transaction processing that will appear transparent to users. One recent standard,

16   Extensible Markup Language (XML), was adopted by the World Wide Web Consortium

17   in February, 1998. In its broadest sense, XML is a system for defining, validating, and

18   sharing document formats on the Web, providing a universal format for structured

19   documents and data. XML is a markup language for presenting documents on the Web

20   that relies on tags and is a meta-language for defining specific subject matter domains of

21   markup tags. XML stores the definitions of tags in files called Document Type

22   Definitions (DTDs). DTDs, also referred to as dictionaries, vocabularies, or schemas,

1   serve as a uniform source of data definitions for specific industries or fields of

2   knowledge, making it easier to exchange data not only within an organization but also

3   among different companies. XML is an extensible standard because users may define

4   their own electronic document type in the form of a DTD. The simple syntax makes an

5   XML document easy to process by machine while the tags promote human understanding

6   of document contents. XML style sheets, called XSL, describe how the tagged data in an

7   XML program should be displayed. Further information about XML and the World

8   Wide Web Consortium, also known as W3C, can be found at the W3C Web site,

9   http://www.w3c.org.

10   Several efforts underway to standardize transaction processing use XML. In the

11   financial industry, for example, J.P. Morgan & Co. Inc. and Price Waterhouse Coopers

12   recently proposed an XML dictionary called FpML (Financial products Markup

13   Language), which would standardize XML tags in areas such as fixed income derivatives

14   and foreign currency exchange. BizTalk is an industry initiative started by Microsoft

15   Corporation of Redmond Washington to establish a community of standards users with

16   the goal of driving the rapid, consistent adoption of XML to enable electronic commerce

17   and application integration. The BizTalk design emphasis is to leverage existing

18   applications, data models, solutions, and application infrastructure, and adapt these for

19   electronic commerce through the use of XML. The group is defining the BizTalk

20   Framework™, a set of guidelines for how to publish schemas in XML and how to use

21   XML messages to easily integrate software programs together in order to build new

1 solutions. Additional information about the BizTalk Framework is available at

2 http://www.biztalk.org.

3 The Internet Open Trading Protocol (IOTP) provides an interoperable framework

4 for Internet commerce that is independent of the particular type of payment system used

5 and is optimized for the case where the buyer and the merchant do not have a prior

6 acquaintance. IOTP describes the content, format and sequences of messages that pass

7 among the participants, referred to as Trading Roles, in an electronic trade. IOTP defines

8 five different types of Trading Roles (Consumer, Merchant, Payment Handler, Delivery

9 Handler, and Merchant Customer Care Provider) that are the ways in which organizations

10 can participate in a trade. The IOTP framework is centered on an IOTP Transaction that

11 involves one or more organizations playing a Trading Role, and a set of Trading

12 Exchanges. Each Trading Exchange involves the exchange of data, between Trading

13 Roles, in the form of a set of IOTP Messages. Each IOTP Message is the outermost

14 wrapper for an XML document that is sent between Trading Roles that take part in a

15 trade. An IOTP message is a well-formed XML document that contains several

16 components including a collection of IOTP Trading Blocks (Request, Exchange,

17 Response) that carries the data required to carry out an IOTP Transaction. An IOTP

18 Trading Exchange consists of the exchange, between two Trading Roles, of a sequence of

19 documents consisting of three main parts: the sending of a Request Block by one Trading

20 Role (the initiator) to another Trading Role (the recipient), the optional exchange of one

21 or more Exchange Blocks between the recipient and the initiator, and the sending of a

22 Response Block to the initiator by the Trading Role that received the Request Block. For

1   more information regarding IOTP, the reader is referred to an Internet-Draft document

2   describing Version 1.0 of the IOTP, published by the Internet Engineering Task Force

3   (IETF) and available at the IETF web site, http://www.ietf.org , as of February, 2000.

4   The Open Buying on the Internet (OBI, http://www.openbuy.org) standard from

5   the OBI Consortium aims to standardize and secure the corporate purchasing model,

6   especially the high-volume, low-dollar transactions that account for 80% of most

7   organizations' purchasing activities. OBI's goal is to establish a common ground for

8   what is referred to as "The Trading Web," where OBI standards adopters establish trading

9   relationships with other OBI standards adopters through secured access to extranet

10   facilities connected via the Internet, forming dynamic sets of interoperable systems. OBI

11   defines an architectural approach for e-commerce systems, detailed technical

12   specifications, guidelines for development, record layout formats, file formats,

13   communication structures and protocols, compliance testing guidelines, and

14   implementation assistance. The OBI standard includes precise technical specifications

15   for the security, transport, and contents of OBI Order Requests and OBI Orders. In the

16   currently published standard, contents of OBI Order Requests and OBI Orders are based

17   on the ANSI ASC X.12's 850, a standard for an EDI purchase order. The OBI

18   Consortium may provide support for XML documents in the future. For a complete

19   discussion of the OBI technical specifications, consult version 2.0 of the Open Buying on

20   the Internet standard available at www.openbuy.org/obi/specs/obiv2.html .

1   RosettaNet is an initiative by a consortium of more than thirty companies in the

2   personal computer (PC) industry, ranging from manufacturers to resellers. Two XML

3   data dictionaries in development will provide a common set of properties required for

4   conducting business among Consortium members. The first is a technical properties

5   dictionary (technical specifications for all product categories), and the second is a

6   business properties dictionary which includes catalog properties, partner properties (i.e.,

7   attributes used to describe supply chain partner companies) and business transaction

8   properties. The goal is a common business language that will link the entire PC

9   industry's supply chain. These dictionaries, coupled with the RosettaNet Implementation

10   Framework (RNIF, an exchange protocol), form the basis for an e-commerce dialog

11   known as the Partner Interface Process or PIP. RosettaNet's PIPs are specialized system-

12   to-system XML-based dialogs that define how business processes are conducted between

13   electronic component and information technology products manufacturers, software

14   publishers, distributors, resellers and corporate end users. The purpose of each PIP is to

15   enable the development of interoperable applications by providing common business/data

16   models and documents that enable system developers to implement RosettaNet interfaces.

17   Each PIP includes one or more XML documents based on Implementation Framework

18   DTDs, specifying one or more PIP services, transactions, and messages. For further

19   information the reader is referred to the RNIF document designated as version 1.1 and

20   published 11/8/99, discussing the RNIF in detail, available at More information about

21   RosettaNet is available at the Web site, http://www.rosettanet.org.

1   Private vendors, such as Ariba Technologies Inc., Commerce One Inc., and

2   Concur Technologies Inc., are using XML to simplify the process of matching up RFPs

3   and purchase orders over the Web. The Ariba Network platform also provides a range of

4   Internet services for buying and selling organizations, including supplier directories,

5   supplier catalog and content management, access to supplier content, and secure

6   transaction routing. The Ariba Network platform is built around a multi-protocol

7   architecture that allows buyers to send transactions from their Ariba buyer-enablement

8   application in one standard format. The Ariba Network platform then automatically

9   converts the order into the suppliers' preferred transaction protocol, eliminating the need

10   for a single standard for electronic commerce and giving suppliers the freedom to transact

11   in their preferred protocol over the Internet. Ariba Network automatically routes and

12   translates transactions between buying organizations and suppliers using many major e-

13   commerce standards, including Internet Electronic Data Interchange (EDI), VAN-based

14   EDI, Open Buying on the Internet (OBI), secure HTML, e-mail, auto-FAX, Catalog

15   Interchange Format (CIF), and a protocol known as Commerce XML (cXML). cXML

16   defines a set of XML DTDs to describe the characteristics of non-production

17   Maintenance, Repair, and Operations (MRO) goods and services. cXML serves as a

18   meta-language to enable the development of "intelligent shopping agents" to assist with

19   the corporate purchasing function. cXML's request/response messaging is used to

20   exchange transaction data between parties. These messages provide support for purchase

21   orders, charge orders, acknowledgements, status updating, shipment notifications, and

22   payment transactions.

1   The public and proprietary efforts underway to standardize transaction processing

2   in the distributed network environment are largely directed to specific industry, function

3   or subject matter domains, such as PC supply-chain management, financial payment

4   handling, or corporate purchasing. Thus, it appears that the standardization effort is

5   directed to establishing predetermined descriptions of transaction message exchanges or

6   dialogs that are specific to and optimized for a specific subject matter or industry domain.

7   Automated commerce solutions that define interactions in terms of fixed message

8   exchanges forgo the flexibility and adaptability required in today's dynamic

9   marketplaces. There will be a wide range of interactions between any two parties in the

10   marketplace that simply do not lend themselves to easy categorization or definition, and

11   that will change over time as the business needs change and as their relationship changes.

12   XML and related data representation standardization efforts, combined with

13   industry-based e-commerce standards efforts, clearly expand the reach of Internet-based

14   e-business to a wider range of enterprises and are efforts in the direction of an integrated

15   Internet e-commerce environment. But these efforts alone fall short of the complete

16   integration needed. What is needed is a transaction processing architecture that directly

17   supports users' needs in the marketplace and a uniform, consistent and flexible

18   transaction definition capability that supports a full range of transaction processing in a

19   distributed network environment.

1   **SUMMARY OF THE INVENTION**

2   The present invention is premised on the observation that a distributed network

3   marketplace must be able to provide both services and support processes to the parties

4   (users) who participate in the marketplace. For example, a distributor who sells items

5   from a catalog benefits from an easy-to-use catalog update service from its suppliers, and

6   a manufacturer benefits from the ability to request a bid with precise terms and to receive

7   only those responses that meet the specified terms. Automated commerce solutions

8   should allow for flexible and adaptable definition of these types of interactions to

9   promote and facilitate dynamic marketplaces. Thus, the present invention is premised on

10   the further observation that a comprehensive e-commerce solution must provide a

11   framework, or architecture, that allows for the definitions of the most complex

12   interactions between parties to be both easily configured and easily changed by the

13   parties as their business needs change. Such a solution should also be platform

14   independent to support a wide variety of computing environments.

15   The present invention provides a transaction processing architecture for a process

16   automation application, referred to as a commerce exchange server. The transaction

17   processing architecture is premised on a user-centric view in which a transaction is a

18   single unit of work from the perspective of the requesting application, or client. The

19   transaction may require several processing components to achieve its end result.

20   However, once the user defines those components and their process flow using a unique

21   and novel transaction definition data structure, the commerce exchange server produces

1   the messages needed to perform the transaction and manages the message flow to and

2   from the service applications without further intervention from the user. Thus, the

3   commerce exchange server is much more than a mere conduit for the message exchange

4   between client and service applications.

5   In this transaction processing model, every transaction has one and only one input

6   document sent from a requesting client application and one and only one output

7   transaction response document sent back to the client. However, each input and output

8   document may have multiple components, or sub-documents. This design precept

9   provides distinct and significant advantages over other transaction processing solutions.

10   First, it considerably simplifies the design of the commerce exchange server by limiting

11   the message exchange between requesting and service applications. Developing client

12   applications becomes straightforward when the client merely issues a transaction request

13   and gets a single response back with the output it requested. In addition, the commerce

14   exchange server takes the complexities of managing a complete transaction away from

15   the requesting client, moving the low-level transaction processing logic common to all

16   transactions to a single source.

17   Finally, this transaction processing model supports the three most common types

18   of application interaction models in the e-commerce environment. These models are

19   generally known as request/reply, publish/subscribe and broadcast. In an illustrated

20   implementation of the commerce exchange server, the request/reply interaction model

21   allows two parties to exchange information in an asynchronous, or non-blocking, fashion.

1   In asynchronous messaging, the requesting application sends a transaction request to the

2   commerce exchange server and may continue its own processing, without waiting for the

3   transaction to be processed. An acknowledgement response is sent that contains tracking

4   information that allows the requesting party to query the status of the transaction request.

5   In a publish/subscribe interaction model, two applications interact via an intermediary

6   party. The applications that are interested in specific information register with the

7   intermediary party. The information generating application posts, or publishes, the

8   information to the intermediary, which in turn passes this to the registered parties. In this

9   model, the information requestor and the information supplier never interact directly.

10   The broadcast model is a special case of a model known as the multicast model, both of

11   which send a message to the members of a group of parties who support the requested

12   operation. When the group size is less than the entire membership of a domain, a

13   message is broadcast to the group; when the group size equals the entire membership,

14   sending the message to the entire group is referred to as multicasting. The message sent

15   in this type of interaction model is typically one of two types: a request message,

16   resulting in a reply message returned, or a notify message that simply reports information

17   or events. Note also that in the multicast interaction model, the recipient group may or

18   may not be subscription based. The information receiver application determines this

19   from the content of the broadcast message. The transaction model of the present

20   invention provides support for all three interaction models.

21   The transaction processing architecture is further premised on the discovery of a

22   novel transaction definition data structure. This data structure allows the user to define a

1   transaction composed of component operations and to define the order of those

2   operations, including determining whether an operation is a broadcast operation or

3   whether more than one operation should be performed concurrently before proceeding to

4   a next operation. The data structure also allows the user to specify the source of input

5   data needed to perform each operation and to place conditional logic on the execution of

6   an operation, based on results of one or more previously executed operations. The

7   transaction definition data structure allows a transaction to be specifically customized to

8   the business needs of the user who defines the transaction.

9   In an illustrated embodiment of the present invention, the transaction definition

10 data structure is an XML document that includes multiple OPERATION sections for

11 specifying the component operations that make up the transaction. An input section,

12 referred to as a JOIN section, within an OPERATION section of the XML document

13 includes markup tags for specifying the source of input data needed. A conditional logic

14 section, referred to as a SPLIT section, within an OPERATION section of the XML

15 document includes markup tags for specifying whether a subsequent operation in the

16 operation flow should be conditioned on the output of a previous operation.

17   The transaction processing architecture supports this flexible and adaptable

18 transaction definition model. The user provides a unique transaction identifier for each

19 transaction definition, stores them in a database of definitions, and then simply requests

20 that a transaction be performed by its transaction identifier. The transaction processing

21 architecture of the present invention defines a transaction service that performs several

1   essential functions. The transaction service obtains the appropriate definition, builds an

2   internal transaction processing data structure and performs the transaction. In an

3   illustrated implementation of the transaction service, all transaction definitions stored in

4   the database are loaded at start-up of the commerce exchange server, and the transaction

5   service obtains the appropriate definition from memory. In an alternate implementation

6   the transaction service may retrieve the appropriate definition directly from the database.

7   In the illustrated embodiment described herein, the internal transaction processing data

8   structure, referred to as a transaction instance, is a directed acyclic graph (DAG) with the

9   conditionals and mapping functions and logic to represent the definition of the

10 transaction. The transaction service then creates and maps the XML documents with

11 input and output variables in order to create and send the various messages needed for

12 transaction execution. The transactions service evaluates the conditional logic and

13 traverses through the DAG in order to execute the transaction, and produces and sends an

14 output response to the requesting application.

15   The commerce exchange server, including the transaction processing architecture

16 that makes use of the novel transaction definition data structure, may be implemented in

17 any type of distributed network of processor-controlled machines such as, for example, in

18 the Internet environment. Protocols for implementing message exchanges in the Internet

19 environment are disclosed in the Protocol patent application referenced above.

20   Therefore, in accordance with one aspect of the present invention, there is

21 provided an XML (extensible markup language) transaction definition document stored

1   on a computer-readable medium comprising a plurality of operation data portions each

2   defining an operation. The plurality of operations collectively define a transaction. Each

3   operation data portion, when parsed by a process automation application, causes the

4   process automation application to communicate with a service application program to

5   perform the operation. At least one operation data portion comprises a conditional logic

6   data portion that, when parsed by the process automation application, causes the process

7   automation application to condition performance of a next operation on evaluation of

8   operation response data from performing the operation.

9   In another aspect of the invention, at least one operation data portion included in

10   the XML transaction definition document indicates a broadcast operation and includes a

11   broadcast data portion. When parsed by the process automation application, the

12   broadcast data portion causes the process automation application to communicate with a

13   plurality of service applications to cause each service application to perform the

14   operation. In a further aspect of the invention, the broadcast data portion further includes

15   an expression data portion indicating at least one of a mathematical expression, a

16   function, and a variable data item. When parsed by the process automation application,

17   the expression data portion causes the process automation application to evaluate the at

18   least one of the mathematical expression, the function, and the variable data item using

19   the operation response data to determine the success or failure outcome of the broadcast

20   operation.

1   In another aspect of the present invention, there is provided a transaction

2   definition data structure stored on a computer-readable medium comprising a plurality of

3   operation data portions indicating a plurality of operations collectively defining a

4   transaction. Each operation data portion defines an operation. Each operation data

5   portion comprises an operation identifier uniquely identifying the operation among the

6   plurality of operations, a service application name indicating a service application for

7   performing the operation, an input data portion indicating input data used by the service

8   operation for performing the operation, and a conditional logic data portion indicating

9   evaluation data conditioning performance of the next operation on evaluation of operation

10   response data received from the service application performing the operation.

11   In another aspect of the present invention, there is provided a computer-

12   implemented method for performing a transaction comprising the steps of producing a

13   transaction instance data structure indicating a plurality of operations constituting a

14   transaction. The transaction instance data structure indicates a linking of the plurality of

15   operations to indicate an operation performance order. The transaction instance data

16   structure further indicates conditioning logic data for changing the operation performance

17   order such that the plurality of operations is capable of being performed in more than one

18   possible order. The computer-implemented method for performing a transaction further

19   includes, for each of the plurality of operations, producing an operation request message

20   indicating input data for performing an operation, sending the operation request message

21   to a service application to perform the operation using the input data, receiving an

22   operation response message from the service application indicating output data from the

1   operation, and determining a next operation to perform using the conditioning logic data

2   and the output data of the operation response message.

3   In yet another aspect of the present invention, there is provided a distributed

4   transaction processing system comprising a plurality of service application programs each

5   capable of performing an operation, and a data store including a plurality of transaction

6   definitions. Each transaction definition indicates a transaction definition name uniquely

7   identifying the transaction definition and a plurality of operation definitions indicating a

8   plurality of operations constituting a transaction. The distributed transaction processing

9   system further comprises a requesting application program that produces a transaction

10   request message indicating a transaction definition name identifying one of the plurality

11   of transaction definitions included in the data store, and a computer having a memory

12   device for storing a process automation application. The process automation application

13   receives the transaction request message indicating the transaction definition name from

14   the receiving application program and uses the transaction definition name to obtain the

15   transaction definition from the data store. The process automation application produces

16   an operation request message for each operation definition included in the plurality of

17   operation definitions and sends the operation request messages to at least one service

18   application program. The at least one service application program sends an operation

19   response message to the process automation application in response to receiving an

20   operation request message. The process automation application produces a transaction

21   response message using the operation response messages, and sends the transaction

22   response message to the requesting application.

1   The novel features that are considered characteristic of the present invention are

2   particularly and specifically set forth in the appended claims. The invention itself,

3   however, both as to its organization and method of operation, together with its

4   advantages, will best be understood from the following description of an illustrated

5   embodiment when read in connection with the accompanying drawings. In the Figures,

6   the same numbers have been used to denote the same component parts or steps.

7   **BRIEF DESCRIPTION OF THE DRAWINGS**

8   FIG. 1   is a block diagram schematically illustrating a systems architecture for

9   managing transaction message flow in a distributed computer network according to the

10   present invention;

11   FIG. 2   is a block diagram showing the types of messages and the general

12   message flow of a transaction between the transaction service component and the service

13   and requesting applications of FIG. 1;

14   FIG. 3   schematically illustrates the inputs to the transaction service function of

15   producing an operation request document according to an illustrated embodiment of the

16   invention;

17   FIG. 4   schematically illustrates the inputs to the transaction service function of

18   producing an operation response document according to an illustrated embodiment of the

19   invention;

1   FIG. 5   is a block diagram schematically illustrating the major entities of the

2   transaction data structure and their organization;

3   FIG. 6   schematically illustrates a first portion of the DTD of the XML

4   document that functions as the transaction data structure according to an illustrated

5   implementation of the present invention;

6   FIG. 7   schematically illustrates a second portion of the DTD of the XML

7   document that functions as the transaction data structure according to an illustrated

8   implementation of the present invention;

9   FIG. 8   is a flowchart illustrating transaction processing as performed by the

10   transaction service of FIG. 2 and using the transaction definition data structure of FIG. 5,

11   according to an illustrated implementation of the present invention; and

12   FIG. 9   is a simplified block diagram illustrating a distributed computer network

13   including several processor-controlled machines, showing the components of one suitably

14   configured processor-controlled machine in which the present invention may be used, and

15   further illustrating the software product of the present invention and its use in conjunction

16   with a machine in the network.

1   **DETAILED DESCRIPTION OF THE INVENTION**

2   **1.   A commerce server architecture utilizing the transaction data structure.**

3   **a.   Process components.**

4   FIG. 1 illustrates a system architecture for enabling application-to-application

5   interaction in a distributed computer network. Specifically, the system architecture of

6   FIG. 1 illustrates an inter- or intra-enterprise Internet-based electronic commerce

7   architecture including process automation application 10, referred to as a commerce

8   exchange (CX) server. CX server 10 operates as a type of clearinghouse, receiving

9   operation requests posted by client components 34 and 38 and directing them to

10   appropriate service components 26, 28 and 30 identified ("signed up") to CX server 10 as

11   being available to perform those services. In this capacity, much of the processing

12   performed by CX server 10 involves searching for service component by service

13   operation and searching for client components by their identification numbers. CX server

14   10 also performs a variety of administrative functions including transaction tracking and

15   audit functions and disaster recovery functions.

16   Each application component is referred to as a commerce exchange component, or

17   CXC. As shown in FIG. 1, there may be any number of CXCs identified to CX server

18   10. A CXC application either provides one or more services or originates a transaction

19   request, or both. A CXC application may be integrated with CX server 10 as a built-in

20   component residing on the same machine or it may be a third party application resident

21   on a different machine. For example, service application 30 is resident on machine 24

1   and accessible to CX server 10 via communications connection 29, and requesting

2   application 38 is resident on machine 36 and accessible to CX server 10 via

3   communications connection 35. The type of architecture model illustrated in FIG. 1 may

4   be variously described in the literature as an information bus model, a client-server model

5   or a cooperative agent model.

6   i.   Communication Service.

7   CX server 10 includes several processing services: Communication service 12;

8   XML/DOM service 14; Transaction service 200; and Persistence service 19.

9   Communication service 12 provides interfaces for accepting and establishing

10   connections, and sending and receiving messages through various network transport

11   protocols.   In an illustrated implementation of CX server 10, the network transport

12   protocol supported is TCP/IP, but other transport protocols may be supported as well.

13   Communication service 12 also provides a variety of other communications-related

14   services including notification of broken connections, fragmentation and assembly of

15   messages, and connection-level session management and handshaking functions.

16   ii.   Application Interaction Protocol Processing.

17   CX server 10 also includes an application interaction protocol processing function

18   400. CX server 10 is a document-centric process automation application, exchanging

19   messages in the form of XML documents 40 between CXCs. These XML documents

20   form the underlying message exchange protocol, referred to as the Commerce Exchange

21   Interaction Protocol, hereafter CXIP. Standardizing the messaging format in this manner

1   allows for the straightforward integration of third party applications as CXCs, without the

2   absolute requirement for application-specific libraries. Each CXC includes two software

3   interface components (not shown) for extracting transaction data from XML-based

4   message 40. A transportation/communication module handles the syntax and semantics

5   of the Application interaction message received from CX server 10 over the particular

6   communications transport mechanism (e.g., TCP/IP), receiving a message and returning

7   an XML document. Then, an XML/DOM (Document Object Model) module receives the

8   XML document output produced from the transportation/communication module, parses

9   the document and returns one or more DOM objects that are passed to the application

10   logic for handling as standard program objects. The use of DOM objects is discussed in

11   more detail below. A CXIP message is in the data representation format specified by

12   XML, which is presumed to be an 8-bit character format in the present implementation.

13   Sending and receiving applications have the responsibility of encoding and decoding data

14   embedded inside a CXIP message.

15   The present implementation of CXIP supports eight (8) message types that

16   implement the three most common application interaction models (Request/Reply,

17   Publish/Subscribe and Broadcast) in the Internet environment. These eight message

18   types are Request, Reply, Cancel, Publish, Notify, Subscribe, Unsubscribe, and

19   Acknowledge. An Acknowledge message is a special type of message used to

20   acknowledge receipt of all of the other message types. An Acknowledge message may

21   contain any information needed for tracking purposes, such as for querying the status of a

22   prior request, or purposes of establishing an audit trail or transaction log. An application

1   should follow the application interaction protocol by sending an Acknowledge message

2   for each received message, except for the Acknowledge message itself. Application

3   interaction models may be implemented in either synchronous or asynchronous mode.

4   An illustrated implementation of CX server 10 operates in asynchronous mode, also

5   referred to as the offline, or non-blocking, model. The Protocol patent application

6   provides additional details about an illustrated implementation of the application

7   interaction protocol.

8   The basic transport assumption in the application interaction protocol, CXIP, used

9   by CX server 10 is the guaranteed delivery of messages. As long as this requirement is

10   satisfied, the underlying transport protocol may be any standard communications

11   protocol. As noted above, the present implementation of CXIP is based on TCP/IP. In

12   this implementation, CXIP messages are transmitted as TCP data between applications.

13   A field size data item in the fixed-length message header of a CXIP message indicates the

14   length of the associated message content in byte counts so that the receiver may easily

15   determine the end of a message without having to test for a special message-termination

16   character. CXIP may also be implemented on top of other transport mechanisms such as

17   SMTP and FTP. Cooperating applications (CXCs) based on different transportation

18   mechanisms (e.g., SMTP or FTP) are implemented by including a bridging mechanism

19   in Communication service 12 (not shown) for translating messages between TCP/IP and

20   SMTP and FTP message formats. To enable HTTP-based interactions a MIME type may

21   be defined, such as *"application/x-cxip-v10"*, and it is straightforward to develop a

22   browser plug-in to handle CXIP messages.

1   iii.   Transaction Service.

2   Transaction service 200 provides interfaces for working with transaction logic,

3 tracking a transaction thread, traversing transaction logic and performing transaction

4 execution. CX server 10 provides a virtual workspace, or transaction execution space, to

5 participating (registered) CXC applications. A CXC submits a transaction request based

6 on a published CX transaction document type declaration (DTD). Upon receipt of a

7 transaction, CX server 10 identifies the set of operations that comprise the transaction

8 based on a transaction definition in data store 18, and then executes the transaction by

9 providing operation requests to CXCs identified as registered to perform the respective

10 operations. Each invoked CXC performs the specified operation request(s) and sends

11 back results to CX server 10, which, after completion of all operation requests, returns the

12 transaction response back to the originating CXC. A transaction definition takes the form

13 of a directed acyclic graph. CX server 10, with knowledge of the transaction logic from

14 the transaction definition, controls all processing decisions including which operations to

15 perform, to which CXC to forward an operation request, how to process the conditions on

16 the services, which information to pass and receive, and when to terminate processing.

17   iv.   XML/DOM Service.

18   XML/DOM service 14 provides interfaces and services for handling the XML

19 documents 40 that form the basis of the message exchange protocol. Services include

20 parsing and constructing XML documents, and building and accessing DOM (Document

21 Object Module) object trees. The Document Object Model (DOM) is a platform- and

22 language-neutral application programming interface (API) for HTML and XML

1   documents that models these documents using objects. The DOM provides a standard set

2   of objects for representing HTML and XML documents, a standard model of how these

3   objects can be combined, and a standard interface for accessing and manipulating them.

4   As an object model, the DOM identifies the semantics of these interfaces and objects,

5   including both behavior and attributes, and the relationships and collaborations among

6   these interfaces and objects. Because of its platform- and language-independent format,

7   the DOM is used as an interface to proprietary data structures and APIs, instead of

8   product-specific APIs, in order to achieve application interoperability with less effort.

9   Additional information regarding the DOM may be found at http://www.w3.org/DOM/.

10   XML/DOM service 14 may make use of any public domain XML parser.

11   Although the XML-based document messaging format is primarily used for exchanging

12   active messages, some internal data used by CX server 10 are also represented and stored

13   as XML documents. For example, the transaction directed acyclic graph that defines the

14   component services of a transaction is an XML document. Therefore, other service

15   components, such as transaction service 200, may use XML/DOM service 14 for

16   translation between XML syntax and an internal data format requirement.

17   v.   Persistence Service.

18   Persistence service 19 provides interfaces for storing information into and

19   retrieving information from external data stores 18. From the perspective of CX server

20   10 or a CXC, data entering into or coming from data stores 18 are in XML document

21   format. Persistence service 19 has the responsibility of mapping between an XML

1   document and the respective data store schema. In an illustrated implementation of CX

2   server 10, data stores 18 include a Netscape™ message server, a Netscape™ LDAP

3   server, and an Oracle™ database server. Support for flat files is also possible. Examples

4   of information that are included in data stores 18 are system parameters, events and alerts,

5   and transaction definitions.

6   **b.   Process and Threading models.**

7   CX server 10 executes as a single process that listens to one listener port and one

8   administrative port for application protocol (CXIP) messages. The single process model

9   distinguishes CX server 10 from conventional application servers that follow the

10   traditional multi-process model. The single process model is critical to the

11   implementation of conditional-logic transaction processing and the complexities of event

12   notification and process control over the CXCs. Moreover, the single process model

13   simplifies administration of the CX server 10 by the system administrator, and is more

14   efficient in database access than a multi-process model. In addition, a single multi-

15   threaded process is typically more efficient than multiple single or multi-threaded

16   processes because it uses fewer system resources such as memory and disk space,

17   assuming that each thread is scheduled as having the same priority as the kernel thread.

18   The capability of deploying a backup CX server addresses the problem of a single point

19   of failure caused by using a single process model.

20   CX server 10 supports both a single-thread and multi-thread model. A single-

21   threaded CX server listens to both the administrative and listener ports at the same time

1   and processes incoming request one after another, in serial fashion. Priority processing is

2   not supported and event processing support is restricted. The single-thread model does

3   not allow for the CX server to load CXC libraries. The multi-threaded CX server uses

4   multiple threads for listening and accepting connections from the administrative port,

5   listening and accepting connections from the listener port, listening and receiving

6   messages from established connections, priority processing of transactions (messages),

7   and executing CXC libraries loaded as part of the process. The multi-threaded model

8   supports both serial and non-serial processing of requests. Serial and non-serial

9   processing are distinguished by whether the message listening thread waits for

10   termination of the thread that is created to process the message. Threading and

11   serialization are determined by configuration parameters provided at startup.

12   In one embodiment of CX server 10, a commerce exchange component (CXC) is

13   expected to establish a persistent connection, throughout the lifetime of the CXC, to the

14   CX server and to use the connection for all message exchanges. The CX server uses the

15   connection to determine the existence of the CXC in the network. Each message received

16   through a persistent connection is processed concurrently using an independent thread.

17   This implementation improves message-processing performance, minimizes the usage of

18   system resources, and eliminates the overhead of establishing and terminating a

19   connection for each new request.

20   In the illustrated implementation of CX server 10 herein, CX server 10 supports

21   asynchronous transaction processing. That is, when an operation request is sent from CX

1   server 10 to a CXC, the processing thread does not block for a response from the CXC

2   and instead sets the state of the transaction and exits from the thread. When a response

3   message is received, the transaction is executed based on the state and the type of

4   response. Support for asynchronous transaction processing achieves efficiency from the

5   single shared connection between CX server 10 and a CXC. Requests may be sent from

6   the CX server simultaneously in multiple threads and the responses may be returned in

7   any order according to how the CXC process them, without waiting for the requests to be

8   performed serially. In addition, timer events may be introduced more easily, thus

9   creating an event-driven processing model.

10   c.   **Distributed transaction processing support.**

11   FIG. 1 also illustrates a representative configuration of the application architecture

12   required to implement transaction processing in a distributed computer network such as

13   the Internet. This application architecture makes use of the Document Object Model

14   (DOM) described above. Service application 30 and requesting (client) application 38

15   each includes transportation/communication module 44 for handling the syntax and

16   semantics of application interaction message 40 received from CX server 10 over a

17   TCP/IP transport mechanism. Transportation/communication module 44 receives

18   message 40 as TCP/IP data and returns an XML document. In service application 30,

19   XML/DOM module 42 receives the XML document output produced from

20   transportation/communication module 44, parses the document and returns one or more

21   DOM objects that are passed to service application logic 21 for handling as standard

22   program objects. Similarly, in requesting application 38, transportation/communication

1   module 44 receives message 40 as TCP/IP data via communications path 35 and returns

2   an XML document. XML/DOM module 46 then receives the XML document output

3   produced from transportation/communication module 44, parses the document and

4   returns one or more DOM objects that are passed to application logic 37 for handling as

5   standard program objects. This component module application architecture enables any

6   third party application to be straightforwardly integrated as a commerce exchange

7   component (CXC) in the domain of a commerce exchange server. Development of these

8   component modules is technically straightforward in either Java or C++ implementations.

9   CX server 10 also supports distributed transaction processing. A CX server in

10   one enterprise or network may communicate with a CX server in another enterprise or

11   network to cooperatively fulfil transaction requests. Thus, one CX server 10 that cannot

12   fulfil a service component of a transaction request using a participating CXC in its own

13   domain may send the operation request to another CX server (not shown in FIG. 1) that

14   includes a participating CXC that has the capability to perform the service. This feature

15   enables an enterprise or group of enterprises to deploy cooperating commerce exchange

16   applications. Note also that, while FIG. 1 shows TCP/IP as the message transport

17   protocol, transportation module 44 may be implemented on top of SMTP or FTP as well.

18   Cooperating applications (CXCs) based on different transportation mechanisms may also

19   be implemented by developing a bridge that translates messages from one protocol to

20   another.

1 **2. Transaction message types and message flow.**

2 Preliminary to describing transaction processing and its associated data structures,

3 definitions are provided for some terminology that has specific meanings in the context of

4 the present invention. These terms have the meanings given here throughout this

5 disclosure, rather than any meanings that may occur in other sources, such as, for

6 example, in documents, if any, that are incorporated by reference herein elsewhere in this

7 description.

8 The term *data* or *data item* refers herein to physical signals that indicate or

9 include information. Data includes data existing is any physical form, and includes data

10 this is transitory or is being stored or transmitted. For example, data could exist as an

11 electromagnetic or other transmitted signal or as a signal stored in electronic, magnetic or

12 other form. A *data structure* as used herein is any combination of interrelated data items.

13 For example, an XML document is a data structure. A data item *indicates* a thing, an

14 event or a characteristic when the item has a value that depends on the existence or

15 occurrence or the measure of the thing, event or characteristic. A first item of data

16 indicates a second item of data when the second item of data can be obtained from the

17 first item of data, when the second item of data can be accessible using the first item of

18 data, when the second item of data can be obtained by decoding the first item of data, or

19 when the first item of data can be an identifier of the second item of data.

20 An *operation* is a single, atomic process that acts upon input data to achieve a unit

21 level function. An operation may sometimes be referred to as a *service*. The CX server

1   handles an operation as a single unitary process, while the scope and nature of the

2   processing involved in an operation is defined by the service application that performs the

3   operation. A *transaction* is a set of one or more operations that are to be performed in a

4   defined order under given conditions by one or more participating service applications.

5   A *transaction definition* is a data structure that defines a type or category of valid

6   transaction to CX server 10. A transaction definition includes the component operations

7   that constitute the transaction, the identity of the input data items required to perform

8   each operation and the source of values for that data. A transaction definition also

9   includes process flow information that indicates conditional logic, if any, to be applied to

10   a component operation, and the data items and format of the output results of the

11   transaction. Note that a transaction definition may include only one transaction. A

12   *transaction database* is a collection of one or more transaction definitions. Each

13   transaction definition includes a unique identifier within a given domain referred to

14   herein as a *transaction definition name*.

15   Every transaction definition conforms to a *transaction directed acyclic graph* data

16   structure, or *transaction DAG*. That is, the transaction DAG specifies the ordered set of

17   data items that are both required and optional for a transaction definition. A directed

18   acyclic graph is known in the art as a set of nodes and a set of ordered pointers between

19   the nodes that define at least one path through the graph, subject to the constraint that no

20   path starts and ends with the same node.

1   A *transaction instance data structure*, or *transaction instance*, is a specific

2   implementation of a transaction definition that indicates the specific data to be used to

3   perform the transaction defined by the transaction definition.   Thus, a transaction

4   definition may be viewed as providing a template for producing a transaction instance

5   when provided with specific input data on which to operate. A transaction instance has a

6   unique identifier within a given domain, referred to as a *transaction ID*, associated with

7   it.

8   In the illustrated implementation of the transaction service described below, a

9   transaction definition is specified using *Extensible Markup Language*, or *XML*, and so is

10   a data object called an *XML document*. *XML* describes a class of data objects called XML

11   documents and partially describes the behavior of computer programs that process them.

12   XML is an application profile or restricted form of SGML, the Standard Generalized

13   Markup Language [ISO 8879]. By construction, XML documents are conforming SGML

14   documents. Each XML document has both a logical and a physical structure. Physically,

15   the document is composed of units called entities. An entity may refer to other entities to

16   cause their inclusion in the document. A document begins in a "root" or document entity.

17   Logically, the document is composed of declarations, elements, comments, character

18   references, and processing instructions, all of which are indicated in the document by

19   explicit markup declarations. The logical and physical structures must nest properly, as

20   described in "4.3.2 Well-Formed Parsed Entities" in the World Wide Web Consortium

21   XML specification. A software module called an XML processor is used to read XML

1   documents and provide access to their content and structure. It is assumed that an XML

2   processor is doing its work on behalf of another processing entity or module.

3   An XML *document type declaration* contains or points to markup declarations

4   that provide a grammar for a class of documents. This grammar is known as a *document*

5   *type definition*, or *DTD*. The document type declaration can point to an external subset

6   containing markup declarations, or can contain the markup declarations directly in an

7   internal subset, or can do both. The DTD for a document consists of both subsets taken

8   together. An XML document is *valid* if it has an associated DTD and if the document

9   complies with the constraints expressed in its associated DTD. An XML document is a

10   *well-formed XML document* if the document, taken as a whole, matches the XML

11   production labeled "document," meets all the constraints with respect to being well-

12   formed given in the XML specification, and each of the parsed entities referenced directly

13   or indirectly within the document is well-formed. A well-formed XML document may

14   also be valid if it meets additional criteria, as specified in World Wide Web Consortium,

15   *Extensible Markup Language (XML) 1.0: W3C Recommendation* 10-February-1998.)

16   Additional information about XML is available at http://www.w3.org/XML and

17   www.w3.org/TR/PR-xml- 971208.

18   FIG. 2 illustrates the components of an illustrated embodiment of the transaction

19   service architecture and the message types associated with a transaction instance.

20   Transaction service 200 is responsible for producing some of the messages involved in

21   performing a transaction, and for managing the message flow necessary to perform a

1   transaction. FIG. 2 shows the transaction message flow and assumes that messages are

2   received by CX server 10 and, after processing by other components (e.g.,

3   communications service 12 and application interaction protocol processing service 400),

4   are passed to transaction service 200. There are four types of messages managed by

5   transaction service 200. These are a transaction request message, an operation request

6   message, an operation response message, and a transaction response message. Note that

7   in the illustrated embodiment of CX server 10 described herein, each of the four types of

8   messages is an XML document that conforms to the application interaction protocol

9   handled by application interaction protocol processing service 400 (FIG. 1) and described

10   in the Protocol patent application.

11   A requesting (or originating) application 34 submits a *transaction request* 284 to

12   transaction service 200. Transaction request 284 is a data structure that indicates a

13   request to process a transaction according to the transaction definition identified by a

14   transaction definition name included in transaction request 284. A transaction is single

15   unit of work from the perspective of the requesting application, or client. In the

16   transaction processing model of CX server 10, every transaction has one and only one

17   input document and one and only one output document, although each input and output

18   document may have multiple sub-document components. Transaction service 200

19   receives request 284 and uses the transaction definition name to obtain the appropriate

20   transaction definition 282. In the illustrated implementation of transaction service 200,

21   all transaction definitions 280 included in transaction database 296 are loaded into

22   memory at the start-up of CX server 10. However, transaction service 200 could also

1   retrieve the appropriate transaction definition 282 from among all transaction definitions

2   280 included in transaction database 296.

3   Transaction service 200 uses transaction DTD 50, transaction definition 282 and

4   transaction request 284 to produce a transaction instance data structure 270 (FIG. 3). The

5   transaction instance is an internal data structure that transaction service 200 uses to

6   perform the requested transaction. In an illustrated embodiment of transaction service

7   200, the transaction instance data structure is a directed acyclic graph. For every

8   operation included in the transaction instance, transaction service 200 produces an

9   *operation request document* 286. Operation request document 286 is sent to a service

10   application 26 (a CXC) to perform the operation. FIG. 3 schematically shows the

11   production of operation request document 286 using transaction definition 282,

12   transaction request 284, transaction instance 270 and transaction DTD 50. Transaction

13   service 200 uses transaction definition 282 and transaction request 284 to produce

14   transaction instance 270, which includes information about each operation in the

15   transaction. Each operation is uniquely identified within transaction instance 270 and

16   includes the name of the service application that is to perform that operation. Transaction

17   service 200 obtains the input data needed for execution of the named operation from

18   transaction request 284 and provides it in operation request document 286 according to

19   specifications provided in transaction instance 270. Additional information about the

20   content of the operation request document and how it is produced is discussed below.

1   Returning again to FIG. 2, transaction service 200 may send several operation

2   request documents 286 to a single service application 26, and may send operation request

3   documents 286 from a single executable transaction to several service applications 26 and

4   30. When a service application 26 completes an operation, it produces an *operation*

5   *response document* 290 indicating the results of the operation and sends operation

6   response document 290 to transaction service 200. Operations within a transaction

7   instance are performed according to an order specified in transaction definition 282.

8   Thus, transaction service 200 tracks the receipt of operation response documents both to

9   determine what operation(s) to perform next and to determine when a transaction instance

10   is complete. Transaction service 200 may use operation results included in an operation

11   response document 290 to produce a subsequent operation request document 286 for a

12   subsequent operation to be executed.

13   When all operations of a transaction instance have been completed, transaction

14   service 200 produces a transaction response document 294, as shown schematically in

15   FIG. 4, using operation response documents 290, and transaction instance 270.

16   Transaction service 200 obtains from transaction instance 270 the format in which the

17   originating requesting application expects to receive the output results of a completed

18   transaction, and prepares transaction response document 294 using the results provided in

19   operation response documents 290. Then, as shown in FIG. 2, transaction service 200

20   returns transaction response document 294 to requesting (originating) application 34.

1 **3.   Description of the Transaction DAG Data Structure.**

2 **a.   Functional components of a transaction.**

3   Transaction definition 282 of FIG. 2 serves as a template for a specific transaction

4 instance and is an XML document having the logical and physical structure specified by

5 its associated transaction DTD 50. The general organization and major functional entities

6 of the transaction directed acyclic graph data structure 50 are schematically illustrated in

7 FIG. 5, with each functional entity shown as a named rectangular box. The identifying

8 data entity names used in FIG. 5 are not intended to limit the data structure in any way.

9 The data entities are illustrated in a hierarchy to show each entity's constituent parts. An

10 entity that may have more than one occurrence is illustrated by multiple offset boxes.

11 Each occurrence includes all of the entities at lower levels in the hierarchy. Entities that

12 are composed of the same data items are labeled with the same reference numbers. An

13 entity that is required is shown with its box in solid outline while the box of an optional

14 entity is shown with a dark dashed outline. A required entity indicates that either the data

15 is explicitly included in the data structure or the necessary data is obtained from some

16 other source by default. The entities that exist below an optional entity in the hierarchy

17 are shown as being either required or optional for the case when the optional higher level

18 entity is present in the transaction. Because an XML DTD expresses both a logical and a

19 physical structure, some of the data entities have logical processing associated with them.

20 The entities and their processing behaviors are defined as follows. Note that the

21 interpretation of DTD 50 described below are defined by a specific implementation of

22 transaction service 200, and are not associated with the DAG data structure. For

1   example, the default behaviors that are described below when no value is provided for a

2   tag or when an optional section is missing indicate the specific interpretation of the

3   illustrated embodiment described herein. The interpretation of DTD 50 described below

4   thus reflect an illustrated embodiment of transaction service 200, and other interpretations

5   are also possible.

6   A transaction instance is composed of a set of ordered operations 54. In the

7   directed acylic graph, an operation is represented by a node in the graph. Every

8   transaction has two specific nodes, or operations, called the head operation and the tail

9   operation. Operations 61 and 63 are shown as the head and tail operations respectively.

10   Operation flow within a transaction always proceeds from the head operation to the tail

11   operation. There may be one or more operations between the head and tail operations but

12   each operation is performed only once. Note that if the operation is a broadcast

13   operation, it is still considered to be performed only once, even though the operation may

14   be sent to many service applications to be performed. There may be more than one

15   possible path through the graph from the head operation to the tail operation, and one of

16   those possible paths is executed at runtime. Thus, the path through the graph for a given

17   transaction definition will not necessarily be the same for each transaction instance of that

18   transaction definition because of differing run time conditions.

19   Each operation is defined to include five functional entities: name 55, Service

20   application name 57, INPUT 60, CONDITIONAL LOGIC 58 and OPERATION

21   LINK(S) 56. These entities provide the information used to produce the operation

1 request document 286 used by a service application to perform this operation and to

2 provide conditional logic to determine which operation(s) is to be performed after the

3 completion of this operation. A unique operation name 55 represents the operation

4 within the context of its transaction. A service application, or CXC, name 57 specifies

5 the service application that can perform this operation. Note that the name of the

6 operation can be determined at run-time by looking up a CXC which has signed up to

7 perform that operation.

8   The INPUT entity, which is the only required entity, provides information

9 sufficient to prepare the operation's operation request document 286 (FIG. 2 and FIG. 3).

10 A list of expressions 60, referred to as INPUT entity logic, is used to build the input

11 arguments for the operation request message 286 for the operation to which this INPUT

12 entity belongs. If the processing for the INPUT entity fails, the operation will not be

13 executed. If no INPUT entity is provided, a default INPUT consolidates all of the

14 preceding operations' operation response documents 290 into the operation request

15 document 284 for this operation. The ARGUMENT component 66 provides an argument

16 to add to the operation request document 286 for this operation. It specifies the name and

17 type of the argument along with a tag indicating if it is required or optional. The

18 associated EXPRRESSION component 70 defines how to derive the value for this

19 argument. An argument may derive its input data from another document, or generate a

20 value based on some EXPRESSION. DOCUMENT component 72 identifies an XML

21 document and defines how that document should be mapped to the indicated argument

22 (ARG). It defines the operation that contains the document and the relevant section(s) of

1   the document to extract. Transaction DAG structure 50 also includes runtime data in the

2   form of OUTPUT entity 59 which includes DOCUMENT entity 73, for use in assembling

3   the output response document.

4   An important feature of DAG structure 50, and the reason that there may be more

5   than one possible path through a transaction instance graph, is that the execution of one or

6   more of OPERATIONS 54 (except for operations 61 and 63) may be conditioned on the

7   output of previous operations. The OPERATION LINK((S) component 56 refers to

8   explicit links between the present (source) operation and a destination (next) operation.

9   Whenever the operation identified as the source operation completes, the operation

10   identified as the destination operation is considered for possible execution. Evaluation

11   for execution is accomplished using the CONDITIONAL LOGIC (CL) entity 58. CL

12   entity 58 is used to decide which operations to consider for execution whenever the

13   operation to which the CL entity belongs completes execution. It is comprised of a series

14   of statements (STMT) 64 that include an EXPRESSION entity 70 which is evaluated,

15   typically using the output results of the completed operation. For each statement that

16   evaluates to a true condition, the list of operations held by that statement in the

17   OPERATION ID entity 74 is returned for consideration for possible execution. If there is

18   no CL entity for an operation, all operations identified as destination operations in the

19   OPERATIONS LINK entity will be considered for possible execution.

20   An optional entity in transaction DAG structure 50 is the BROADCAST entity

21   62. The presence of BROADCAST entity 62 indicates that the operation is a broadcast

1   operation and should be sent to more than one service application for processing. The

2   optional subsections place success criteria on the broadcast operation to determine when

3   to advance the operation as a whole. If RESPONSE entity 68 is present, a data value

4   indicates that there are a minimum number of successful responses expected before this

5   operation may be advanced. If EXPRESSION entity 70 is present, it specifies that an

6   action should be performed and a value returned and evaluated before this operation may

7   be advanced. An expression can be a simple value, a math operation, the value of a

8   variable, or the return value of a function.

9   **b.   An illustrated implementation of a Transaction DAG.**

10   In an illustrated embodiment of the present invention, the transaction DAG data

11   structure has the structure of the document type definition (DTD) shown in Table 1 and

12   illustrated in FIG. 6 and FIG. 7. The INPUT entity 60, CONDITIONAL LOGIC entity

13   58 and OPERATION LINK(S) entity 56 of FIG. 5 are referred to as JOIN section 87,

14   SPLIT section 86, and OPLINK section 85, respectively, in Table 1 and in FIG. 6 and

15   FIG. 7.

16   TABLE 1

```
<!DOCTYPE CXTXDAG    [
<!ELEMENT CXTXDAG     (TRANSACTION)*>
<!ATTLIST CXTXDAG
       NAME        CDATA               #REQUIRED
       VERSION     (1.0 | 2.0 |…)      \"1.0\"
       OBJMODEL    (ECXpert | …)       \"ECXpert\"
          >
<!ELEMENT TRANSACTION  (OPERATION)*>
<!ATTLIST TRANSACTION
       NAME        CDATA       #REQUIRED
       TIMEOUT     CDATA       #IMPLIED
       SAVE        CDATA       #IMPLIED
   >
```

```
<!ELEMENT OPERATION     (OPLINK*| SPLIT | JOIN)>
<!ATTLIST OPERATION
      OPID       CDATA         #REQUIRED
      NAME       CDATA         #REQUIRED
      CXCNAME    CDATA         #REQUIRED
      BROADCAST  CDATA         #IMPLIED
      TIMEOUT    CDATA         #IMPLIED
>
<!ELEMENT OPLINK      EMPTY>
<!ATTLIST OPLINK
      SRCOPID    CDATA         #REQUIRED
      DSTOPID    CDATA         #REQUIRED
>
<!ELEMENT SPLIT         (STMT)*>
<!ATTLIST SPLIT
      NAME       CDATA         #REQUIRED
>
<!ELEMENT STMT          (EXPR, OPID+)*>
<!ATTLIST STMT
      NAME       CDATA         #REQUIRED
>
<!ELEMENT EXPR      (VALUE | OPERATOR | VAR |
                                    FUNCTION)*>
<!ATTLIST EXPR
      NAME       CDATA         #REQUIRED
>
<!ELEMENT OPID          EMPTY>
<!ATTLIST OPID
      ID         CDATA         #REQUIRED
>
<!ELEMENT OPERATOR  (VALUE | OPERATOR | VAR |
                                    FUNCTION)*>
<!ATTLIST OPERATOR
      MATHOP     CDATA         #REQUIRED
>
<!ELEMENT VAR         EMPTY>
<!ATTLIST VAR
      VARNAME    CDATA         #REQUIRED
      VALTYPE    CDATA         #IMPLIED
      OPID       CDATA         #IMPLIED
      INPUTDOC   CDATA         #IMPLIED
>
<!ELEMENT VALUE       EMPTY>
<!ATTLIST VALUE
      STRVALUE   CDATA         #IMPLIED
      NUMVALUE   CDATA         #IMPLIED
>
<!ELEMENT FUNCTION      EMPTY>
<!ATTLIST FUNCTION
      LIBNAME    CDATA         #REQUIRED
      FUNCNAME   CDATA         #REQUIRED
      VALTYPE    CDATA         #IMPLIED
```

```
>
<!ELEMENT JOIN          (FUNCTION | ARG*)>
<!ELEMENT ARG           (EXPR)>
<!ATTLIST ARG
        VARNAME    CDATA        #REQUIRED
        VALTYPE    CDATA        #IMPLIED
    >
]>
```

1   c.   **Designing a transaction and specifying an operation.**

2   TRANSACTION section 84 is composed of one or more operations that are to be

3   executed in the order specified under the conditions provided at each juncture. At the

4   transaction level, the designer must provide a NAME for the transaction that must be

5   unique across all transactions that are defined within the domain of CX server 10. The

6   name of the transaction is used for instantiating run-time transactions of this type as the

7   result of a Transaction Request Message 284 (FIG. 2). Optionally, the user may provide

8   a TIMEOUT value for the transaction. If specified, this represents the maximum amount

9   of time, in seconds, that a transaction has to complete execution once it has been started.

10   If no value is provided, or a value of zero is provided, no timeout is assumed and the

11   transaction may take as long as necessary to complete. The user defining the transaction

12   definition may also optionally use the SAVE tag to specify whether or not to save

13   transactions of this type to a data base of transactions (also referred to as the persistence

14   server.) A value of YES for this field indicates that transactions of this type should be

15   saved. If not specified, the default turns saving ON for this transaction type.

16   Examples of portions of transaction definitions are provided to illustrate how the

17   transaction DAG structure is used. Example 1 defines a transaction whose name is

1   "myTransaction" whose run-time instances should not be saved and can take at most 60

2   seconds to execute.

3   Example 1:

```
4        <TRANSACTION NAME="myTransaction" SAVE="NO" TIMEOUT="60">
5        <!— Define operations here -->
6        </TRANSACTION>
```

7   Example 2 defines a transaction whose name is "anotherTransaction" whose run-time

8   instances should be saved and has no maximum execution time restrictions.

9   Example 2:

```
10        <TRANSACTION NAME="anotherTransaction">
11        <!–Define operations here  -->
12        </TRANSACTION>
```

13   TRANSACTION section 84 points to OPERATION section 86. Each operation

14   must specify a NAME, OPID, and CXCNAME, and optionally a TIMEOUT value. The

15   name specifies the logical name of the operation, and will often correspond to the service

16   application (CXC) that executes it. There are two reserved names for two required

17   operations known as CXtsHeadOp and CXtsTailOp. Every transaction must have a

18   CXtsHeadOp and a CxtsTailOp; the CXtsHeadOp is always the first operation in a

19   transaction, and the CXtsTailOp is always the last. These operations are in addition to

20   any operations that are to be included in the transaction. The OPID is a numeric value

21   that must be unique within the transaction definition; no other operation within the

22   transaction may have the same OPID value. The CXCNAME specifies the logical name of

23   the CXC (service application) that can execute this operation. When the CXCNAME

24   indicates a value of '*', any CXC which has registered with CX server 10 as being capable

1 of executing this type of operation may be used. An operation may optionally specify a

2 TIMEOUT value. If specified, this defines the maximum amount of time an operation can

3 take to execute. If no value is provided, or a value of zero is provided, no timeout is

4 assumed and the operation can take as long as necessary to complete. Table 2 provides

5 additional information about each of the data entities in the Transaction and Operation

6 sections of the DAG data structure.

7   TABLE 2: Transaction and Operation Sections

Section Name	Tag Name	Required	Type	Default Value
TRANS-ACTION				
	Meaning: A transaction is a sequence of operations that are to be executed in a defined order under the given conditions.			
	NAME	Yes	Character	None
	Meaning: The name of the transaction. It will be used to refer to the type of transaction for instantiating run-time transactions of this type as the result of a Transaction Request Message. This name must be unique across all defined transactions.			
	TIMEOUT	No	Numeric	Zero
	Meaning: The maximum amount of time the transaction can take to execute, in seconds. If this tag is not present or the value is 0, this means there is no timeout value.			
	SAVE	No	Character	YES
	Meaning: A flag which indicates whether or not to save transactions of this type to the persistence server. If present, any value other than 'YES' or 'yes' will turn saving off for run-time instances of transactions of this type. If not specified, a value of 'YES' is assumed.			
OPERA-TION				
	Meaning: An operation is a request to perform some action. The operation's operation request document is prepared according to the definition provided in the JOIN section. This operation request document is sent to the indicated CXC for an operation request. The resulting message from the CXC is stored as the operation response document.			
	NAME	Yes	Character	None
	Meaning: The name of the operation. This does not have to be unique across the transaction. It refers to the logical name given to this type of operation. There are two pre-defined operation names that must appear in every transaction definition: 'CXtsHeadOp' refers to the first operation in the transaction; 'CXtsTailOp' refers to the last operation in the transaction.			

OPID	Yes	Numeric	None
Meaning: A numeric value assigned to this particular operation. It must be unique within the transaction definition.			
CXCNAME	Yes	Character	None
Meaning:. The name of the CXC which can execute this operation. If the value is an asterisk (*), then the CX will, at run-time, determine the CXC's which can execute this operation (this is often used for broadcast operations). If a specific name is provided, the CX will look for a CXC instance with that name and attempt to have that CXC execute this operation.			
TIMEOUT	No	Numeric	Zero
Meaning:. The maximum time, in seconds, the operation has to complete execution. If no value is provided, or a value of zero is provided, no timeout is assumed.			

1   Example 3 defines a simple operation whose name is "myOperation", has an ID of "1"

2   which can be executed by the service application, "myCXC", and can take at most 60

3   seconds to execute.

4   Example 3:

```
5          <OPERATION NAME="myOperation" OPID="1" CXCNAME="myCXC"
6          TIMEOUT="60 />
```

7   Example 4 defines an operation whose name is "anotherOperation", has an ID of "3",

8   whose CXC should be determined by the CX for each run-time instance, and has no

9   restrictions on the amount of time it can take to execute.

10   Example 4:

```
11         <OPERATION NAME="anotherOperation" OPID="3" CXCNAME="*" />
```

12   OPERATION section 86 points to OPLINK section 85. A link between two

13   operations is specified using an OPLINK section. Each OPLINK section includes a

14   single SRCOPID tag and DSTOPID tag. There can be 1 – n OPLINK sections per

15   operation. The SRCOPID tag is the operation ID of the operation where the link starts,

1   and the DSTOPID tag is the operation ID of the operation where the link ends. Although

2   not required, it is good practice to have the SRCOPID match the OPID of the

3   OPERATION section that contains the OPLINK section.

4   Operation links define the order of execution of the operations. When one

5   operation completes its execution, the operation links specify the set of operations that

6   may be executed as a result. An operation is ready for execution when all other

7   operations that have forward links to this operation have completed execution. Note that

8   this includes operations that are considered complete for any reason, including timeout,

9   failure, or elimination due to split condition logic. If a split condition is specified, the

10   conditions defined by that split condition must be evaluated to determine which of the

11   links to follow. For more details see the section below entitled Specifying conditional

12   operation flow (SPLIT) logic. When defining a transaction, the first set of operations that

13   should be executed must be linked as destination operations from the CXtsHeadOp

14   operation. The final set of operations must all have CXtsTailOp as their destination

15   operations in their operation links. The OPID of the CXtsHeadOp conventionally has a

16   value of zero, although this is not mandatory. Table 3 provides additional information

17   about each of the data entities in the OPLINK section of the DAG data structure.

1

TABLE 3: OPLINK Section

Section Name	Tag Name	Required	Type	Default Value
OPLINK				
	Meaning: Defines a link between the named operations. Whenever the operation identified as the source operation completes, the operation identified as the destination operation is considered for possible execution.			
	SRCOPID	Yes	Numeric	None
	Meaning: The source operation in the link. The value should correspond to the OPID specified in the OPERATION section..			
	DSTOPID	Yes	Numeric	None
	Meaning: The destination operation in the link. This value must correspond to the OPID specified in the OPERATION section of the operation that is the desired destination operation.			

2 Example 5 defines a simple linear transaction graph with three operations, Parse,

3 Translate, and Gateway which are to be executed one after the other. In this example, the

4 Parse operation has an ID of 1, the Translate operation has an ID of 2, and the Gateway

5 operation has an ID of 3.

6 Example 5:

```
7   <OPERATION NAME="CXtsHeadOp" OPID="0" CXCNAME="Head">
8   <OPLINK SRCOPID="0" DSTOPID="1" />
9   </OPERATION>
10  <OPERATION NAME="Parse" OPID="1" CXCNAME="Parse">
11  <OPLINK SRCOPID="1" DSTOPID="2" />
12  </OPERATION>
13  <OPERATION NAME="Translate" OPID="2" CXCNAME="Translate">
14  <OPLINK SRCOPID="2" DSTOPID="3" />
15  </OPERATION>
16  <OPERATION NAME="Gateway" OPID="3" CXCNAME="Gateway">
17  <OPLINK SRCOPID="3" DSTOPID="4" />
18  </OPERATION>
19  <OPERATION NAME="CXtsTailOp" OPID="4" CXCNAME="Tail">
20  </OPERATION>
```

21 Example 6 defines a transaction graph with concurrent operations where the first two

22 operations are executed concurrently, followed by a third operation that is executed after

23 those two complete.

1   Example 6:

```
2            <OPERATION NAME="CXtsHeadOp" OPID="0" CXCNAME="Head">
3            <OPLINK SRCOPID="0" DSTOPID="1" />
4            <OPLINK SRCOPID="0" DSTOPID="2" />
5            </OPERATION>
6            <OPERATION NAME="OP1A" OPID="1" CXCNAME="OP1ACXC">
7            <OPLINK SRCOPID="1" DSTOPID="3" />
8            </OPERATION>
9            <OPERATION NAME="OP1B" OPID="2" CXCNAME="OP1BCXC">
10           <OPLINK SRCOPID="2" DSTOPID="3" />
11           </OPERATION>
12           <OPERATION NAME="OP3" OPID="3" CXCNAME="OP3CXC">
13           <OPLINK SRCOPID="3" DSTOPID="4" />
14           </OPERATION>
15           <OPERATION NAME="CXtsTailOp" OPID="4" CXCNAME="Tail">
16           </OPERATION>
```

17   An operation may be specified as having up to three additional optional

18   components. The operation may be defined as a Broadcast Operation, a set of Split

19   conditions may be specified, and the method of preparing the operation request document

20   to the operation, via a JOIN section, may be defined. Each of these is described in more

21   detail below.

22   **d.   Specifying conditional operation flow (SPLIT) logic.**

23   SPLIT section 86 is used whenever the decision about the next set of operations to

24   execute depends upon some condition. SPLIT section 86 provides the conditional logic

25   on which to base the decision about the operation(s) to consider for execution next, when

26   the operation to which the SPLIT belongs completes execution. SPLIT section 86 is

27   comprised of a set of STMT sections 90 (statement) which contain expressions to be

28   evaluated. Within each STMT section is an expression section (EXPR) 94 and a set of

29   operation ID sections 93 (OPID). An expression is some condition to evaluate; details

30   about specifying an expression are provided in section g. below. The set of operation ID

1   sections 93 indicate those operations that should be slated for execution if the condition

2   specified by the expression evaluates to a true value (a non-zero value). The operation

3   IDs specified in the STMT sections must match one of the operation IDs in the set of

4   operation links (OPLINK) for the operation containing the split condition. If there is no

5   SPLIT section for an operation, all operations identified as destination operations in the

6   OPLINK section will be considered for possible execution. Table 4 provides additional

7   information about each of the data entities in SPLIT section 86 of the DAG data structure.

8   TABLE 4: SPLIT Section

Section Name	Tag Name	Required	Type	Default Value
SPLIT				
	Meaning: The SPLIT is used to decide which operations to consider for execution whenever the operation to which the SPLIT belongs completes execution. It is comprised of a series of statements (STMT) which are evaluated. For each statement that evaluates to a true condition, the list of operations held by that statement is returned for consideration for possible execution. If there is no SPLIT section for an operation, all operations identified as destination operations in the OPLINK section will be considered for possible execution. Any operation listed as part of a statement must also have been listed as a DSTOPID in an OPLINK for the OPERATION where the SPLIT is defined.			
STMT	YES			
	Meaning: A required subsection of a SPLIT section. It consists of an expression (EXPR) to be evaluated and a set of operation ID (OPID) sections listing the IDs of the operations to return if the expression evaluates to true.			
OPID				
	Meaning:. Holds the ID tag to indicate the ID of an operation to be returned if an expression evaluates to true in a split statement.			
	ID	Yes	Numeric	None
	Meaning: The ID of the operation to be returned if the expression in a split statement evaluates to true..			

1   Example 7 defines a split condition in which the next operation should be executed only

2   if the variable PROCEED is included in the output section of this operation's operation

3   response document. (More information about the EXPR section is provided below.)

4   Example 7:

```
5     <OPERATION NAME="MYOP" OPID="1" CXCNAME="MYOPCXC">
6     <OPLINK SRCOPID="1" DSTOPID="2" />
7     <SPLIT>
8     <STMT>
9     <EXPR>
10    <VAR VALTYPE="NUMVALUE" VARNAME="PROCEED" OPID="1"
11    OPVARIOTYPE="OUTPUT"
12    OPDOCIOTYPE="OUTPUT" />
13    </EXPR>
14    <OPID ID="2" />
15    </STMT>
16    </SPLIT>
17    </OPERATION>
18    <OPERATION NAME="ANOTHEROP" OPID="2" CXCNAME="ANOTHERCXC">
19    </OPERATION>
```

20   **e.   Specifying input arguments using the JOIN section.**

21   JOIN section 87 is used to build the input arguments for operation request

22   message 286 (FIG. 2) for the operation to which this JOIN logic belongs. JOIN

23   processing is essentially the translation of one or more documents into the operation

24   request document. The join may be defined to map entire documents (or specific

25   sections) of one or more operations into the operation request document or specific

26   variables. The join may specify that only the input sections be mapped, that only the

27   output section be mapped, or that both the input and output sections be mapped.

28   All operations considered for execution will have their JOIN section executed to

29   build the operation request document. If the JOIN fails, the operation will not be

30   executed. If no JOIN is provided for a particular operation (i.e., no input arguments are

1   specified), the default JOIN will consolidate the operation response documents 290 (FIG.

2   2) of all of the preceding operations that belong to this transaction into the operation

3   request document for this operation. In particular, the default join maps all of the input

4   and output variables of the previous operation(s) into the input and output sections of the

5   operation request document for this operation. As previously noted, default behavior is

6   implementation specific, and the Transaction DAG does not enforce this logic.

7   A join may be comprised of a series of argument sections (ARG) which specify

8   the value to be given to one or more specific named variables in the operation request

9   document of the operation. The ARG section consists of a series of tags defining the

10   properties of the variable(s) in the operation request document and either an expression or

11   document mapping to give the variable its value. Each variable is given a name (NAME)

12   and a representation type (VALTYPE) along with an optional tag indicating if this

13   variable is mandatory or optional (OPTIONAL). If no OPTIONAL tag is provided, or

14   the value of the OPTIONAL tag is "NO" (or "no"), the variable is considered mandatory.

15   If the OPTIONAL tag is present and its value is either "YES" (or "yes"), the variable is

16   considered optional. This is used to determine how to proceed if the named variable

17   cannot be given a value. If an optional variable is not available, the join can still succeed,

18   provided that any other mandatory variables are available. If a mandatory variable is not

19   available, the join is considered to have failed and the operation cannot be executed.

20   Failure of the operation may, in turn, cause the entire transaction to fail. The variables

21   may be given values based on the evaluation of some expression (EXPR) or from a

22   mapped document, or specific section of a document.

1   When a function is to be executed, a user who defines a transaction may specify

2   the JOIN translation in one of several ways. A user may define a dynamic library (DLL)

3   and an entry point; the library is then loaded by CX server 10 and executed. The output

4   of the entry point would be a single document. A user may alternatively provide Java

5   classes for translation purposes, or may define a shell script mechanism. The shell script

6   would get current documents as files and return the expected document in a location

7   expected by CX server 10. Simple document translations may also be defined visually

8   using a user interface mechanism for defining transactions. See Section 5.b. below for

9   additional information about the user interface. Table 5 provides additional information

10   about each of the data entities in JOIN section 87 of the DAG data structure.

11   TABLE 5: JOIN Section

Section Name	Tag Name	Required	Type	Default Value
JOIN				
	Meaning: The JOIN is used to build the input arguments for an operation request message for the operation to which this join belongs. All operations considered for execution will have their JOIN section executed to build the operation request document. If the JOIN fails, the operation will not be executed. If no JOIN is provided, the default JOIN will consolidate all of the preceding operations' operation response documents into the operation request document for the operation.			
	DOCVAR-TYPE	No	Character	DOC
	Meaning: If present in the join, it defines whether the input section (INPUT), output section (OUTPUT), or both sections (DOC) of the documents are to be mapped to the operation request document for the join. This is used for joins where all of the previous operations' documents are to be merged to form the operation request document for the operation to which the join belongs.			
DOC-VAR	Meaning: Defines how the identified XML document should be mapped to the indicated argument (ARG). It defines the operation that contains the document, which of its two documents to use, and the relevant section(s) of the document to extract..			
	OPID	Yes	Numeric	None
	Meaning: The ID of the operation whose document is to be extracted.			

	OPDOCIOTYPE	No	Character	INPUT
	Meaning: Defines whether the operation request document (INPUT) or operation response document (OUTPUT) should be extracted.			
	DOCVAR-TYPE	No	Character	DOC
	Meaning: Defines whether the input section (INPUT), output section (OUTPUT), or both sections (DOC) should be extracted.			
ARG				
	Meaning: An argument to add to the operation request document for this operation. It specifies the name and type of the argument along with a tag indicating if it is required or optional. The associated EXPR defines how to derive the value for this argument.			
	VARNAME	Yes	Character	None
	Meaning: The name of the variable that will be added to the indicated section of the operation request document of this operation. Its value is determined by the expression associated with the argument.			
	VALTYPE	No	Character	NUMVALUE
	Meaning: . The representation of the value. It can be either 'NUMVALUE' for numeric values, 'STRVALUE' for string values, or 'DOCVALUE' for document values.			
	OPTIONAL	No	Character	None
	Meaning:. If present, it indicates if the JOIN argument is required to be in the operation request document or not. If it is not present, it defaults to being required. If a required argument is not present, the JOIN will fail. If an optional argument is not present, the JOIN will proceed.			

1   Example 7 defines a default join for an operation (OPID = 3) which consolidates all of

2   the variables in the preceding operations' operation response documents. In this example,

3   two operations directly precede this operation (OPID = 1 and OPID = 2), and all of the

4   variables in the operation response document for OPID = 1 and OPID = 2 would be

5   mapped into the operation request document for OPID = 3.

6   Example 7:

```
7       <OPERATION NAME="myOP1" OPID="1" CXCNAME="myOPCXC">
8       <OPLINK SRCOPID="1" DSTOPID="3" />
9       </OPERATION>
10      <OPERATION NAME="anotherOP" OPID="2" CXCNAME="anotherOPCXC">
11      <OPLINK SRCOPID="2" DSTOPID="3" />
12      </OPERATION>
13      <OPERATION NAME="thirdOP" OPID="3" CXCNAME="thirdOPCXC">
14      </OPERATION>
```

1   Example 8 defines a join that maps the input variables of the previous operations to this

2   operation's operation request document. In this example, only the variables in the INPUT

3   section of the preceding operations' operation response document are mapped.   The

4   OUTPUT variables would not be mapped.

5   Example 8:

```
6           <OPERATION NAME="myOP1" OPID="1" CXCNAME="myOPCXC">
7           <OPLINK SRCOPID="1" DSTOPID="3" />
8           </OPERATION>
9           <OPERATION NAME="anotherOP" OPID="2" CXCNAME="anotherOPCXC">
10          <OPLINK SRCOPID="2" DSTOPID="3" />
11          </OPERATION>
12          <OPERATION NAME="thirdOP" OPID="3" CXCNAME="thirdOPCXC">
13          <JOIN DOCVARTYPE="INPUTVARLIST">
14          </JOIN>
15          </OPERATION>
```

16   Example 9 defines a join that maps both the input and output variables of previous

17   operations to this operation's operation request document. In this example, the variables

18   in both the INPUT and OUTPUT sections of the preceding operations' operation response

19   documents are mapped.

20   Example 9:

```
21          <OPERATION NAME="myOP1" OPID="1" CXCNAME="myOPCXC">
22          <OPLINK SRCOPID="1" DSTOPID="3" />
23          </OPERATION>
24          <OPERATION NAME="anotherOP" OPID="2" CXCNAME="anotherOPCXC">
25          <OPLINK SRCOPID="2" DSTOPID="3" />
26          </OPERATION>
27          <OPERATION NAME="thirdOP" OPID="3" CXCNAME="thirdCXC">
28          <JOIN DOCVARTYPE="VARLIST">
29          </JOIN>
30          </OPERATION>
```

31   Example 10 defines a join that maps the variables PRICE and QUANTITY from the

32   operation response document of the operation with an ID of 2 to the input variable

- 55 -

1   TOTAL where TOTAL = PRICE * QUANTITY. All of the variables are numeric. The

2   variable, TOTAL, is mandatory.

3   Example 10:

```
4              <OPERATION NAME="MYOP" OPID="2" CXCNAME="MYCXC">
5              <JOIN>
6              <ARG VARNAME="TOTAL" VALTYPE="NUMVALUE">
7              <EXPR>
8              <OPERATOR MATHOP="SUM">
9              <VAR VALTYPE="NUMVALUE" VARNAME="PRICE" OPID="2" />
10             <VAR VALTYPE="NUMVALUE" VARNAME="QUANTITY" OPID="2" />
11             </EXPR>
12             </ARG>
13             </JOIN>
14             </OPERATION>
```

15   Example 11 defines a join that maps a previous operation's operation response document

16   (OPID = 1) as a variable named "OP1OutputDoc." This variable is considered optional.

17   Example 11:

```
18             <OPERATION NAME="MYOP" OPID="2" CXCNAME="MYCXC">
19             <JOIN>
20             <ARG VARNAME="OP1OutputDoc" VALTYPE="DOC" OPTIONAL="YES">
21             <DOCVAR OPID="1" OPDOCIOTYPE="OUTPUT" DOCVARTYPE="DOC" />
22             </ARG>
23             </JOIN>
24             </OPERATION>
```

25   Example 12 defines a join that maps the minimum value of a variable, PRICE, from the

26   operation response documents from a broadcast operation (OPID = 1) as a variable,

27   LowestPrice. This variable is considered mandatory.

28   Example 12:

```
29             <OPERATION NAME="MYOP" OPID="2" CXCNAME="MYCXC">
30             <JOIN>
31             <ARG VARNAME="LowestPrice" VALTYPE="NUMVALUE" OPTIONAL="NO">
32             <EXPR>
33             <VAR VALTYPE="NUMVALUE" VARNAME="PRICE" OPID="1"
34             BCAST_MATHOP="MIN"/>
35             </EXPR>
36             </ARG>
```

1   </JOIN>
2   </OPERATION>

3   **f.   Specifying a broadcast operation.**

4   A broadcast operation is a special type of operation in which more than one

5   instance of the operation is executed. The decision as to how many instances to execute

6   is a run-time decision made by CX server 10 by sending operation requests to every CXC

7   which has registered as capable of executing operations of this type. To specify an

8   operation as a broadcast operation, a BROADCAST section 88 must be included. Within

9   this BROADCAST section, the two optional subsections of RESPONSE 91and EXPR 94

10   may be provided.

11   If the RESPONSE section is present, the MIN tag specifies the minimum number

12   of responses required to be received before advancing past this operation. If no

13   RESPONSE section is defined, the default is that all operation requests sent on behalf of

14   this broadcast operation must be received before the operation as a whole can be

15   advanced.

16   If the EXPR section is present, this indicates an expression that must be evaluated

17   against each response received to determine if it should be counted toward the minimum.

18   If no EXPR section is present, all responses received will be counted toward the minimum

19   (if a minimum is specified).   See the section, Specifying an Expression, for more

20   information about expressions. Table 6 provides additional information about each of the

21   data entities in BROADCAST section 88 of the DAG data structure.

1   TABLE 6: BROADCAST Section

Section Name	Tag Name	Required	Type	Default Value
BROAD-CAST				
	Meaning: If present, indicates the operation is a broadcast operation. The optional subsections can place success criteria on the broadcast operation to determine when to advance the operation as a whole.			
RES-PONSE				
	Meaning: If present, indicates there is a minimum number of successful responses expected before this operation can be advanced.			
	MIN	Yes	Numeric	None
	Meaning: The minimum number of successful responses required for advancement. If present, the value must be greater than zero.			

2   Example 13 defines a broadcast operation where the number of requests is to be

3   determined at run-time. The name of the operation is "myBroadcast" and its ID is 1.

4   Example 13:

```
5          <OPERATION NAME="myBroadcast" OPID="1" CXCNAME="*">
6          <BROADCAST />
7          </OPERATION>
```

8   Example 14 defines a more complex broadcast operation where the minimum number of

9   responses is 1 and the response should contain a variable, PRICE, with a value less than

10   10 to be considered successful.

11   Example 14:

```
12         <OPERATION NAME="myBroadcast" OPID="1" CXCNAME="*">
13         <BROADCAST>
14         <RESPONSE MIN="1" />
15         <EXPR>
16         <OPERATOR MATHOP="LT">
17         <VAR VALTYPE="NUMVALUE" VARNAME="PRICE" OPID="1"
18         OPVARIOTYPE="OUTPUT"
19         OPDOCIOTYPE="OUTPUT"/>
20         <VALUE NUMVALUE="10" />
21         </OPERATOR>
22         </EXPR>
23         </BROADCAST>
24         </OPERATION>
```

1 g.   **Specifying an expression.**

The EXPR section 94 is used for split conditions, for determining values for

3 arguments in a JOIN, and for defining BROADCAST advance criteria. An expression

4 may be a simple value (VALUE), an operation (OPERATOR), a function (FUNCTION),

5 or the value of a specified variable (VAR). When an expression is a simple value,

6 whenever the expression is evaluated, it always returns the configured value. The

7 VALUE parameter of the EXPR section could be used to initialize some variable in an

8 operation to a configured value. An expression may be defined as the value of a named

9 variable using the VAR tag. Whenever the expression is evaluated, the value of the

10 named variable is returned, or an error is returned if the variable could not be found.

11 Using the VAR tag in an expression also requires the following additional information:

12 the name (VARNAME) of the variable; the ID of the operation that contains the variable

13 (OPID); the document that contains the variable (OPDOCIOTYPE); the section of that

14 document that contains the variable (OPDOCVARTYPE); and the representation type of

15 the value (VALTYPE).

16   An expression may be a math operation (OPERATOR) applied to one or more

17 variables. Supported math operations include the most commonly used operations such

18 as greater than, equal to, less than, plus, minus and multiplication. All of the math

19 operations, with the exception of 'PLUS', can only be applied to numeric values

20 (VALTYPE="NUMVALUE"). Note that math operations do not work on document

21 values. An expression may also be defined to return the result of executing a named

22 function. A function as an expression requires the name of the function (FUNCNAME),

1   the path and library where the function is defined (LIBNAME) and the type of value

2   returned by the function (VALTYPE). When evaluated, the EXPR section returns the

3   evaluation result, or an indication of an error if the expression could not be evaluated. An

4   error occurs if the desired variable or function is not available or the expression was not

5   correctly defined. Table 7 provides additional information about each of the data entities

6   in EXPR section 94 of the DAG data structure.

7   TABLE 7: EXPR Section

Section Name	Tag Name	Required	Type	Default Value
EXPR				
	Meaning: A specification to perform some action and return the value. An expression can be a simple value, a math operation, the value of a variable, or the return value of a function.			
VALUE				
	Meaning: The expression should return the specified value whenever it is evaluated.			
	NUMVALUE	Yes	Numeric	None
	Meaning: The actual value to be returned.			
OPER-ATOR				
	Meaning: An action to perform on the indicated variables. With the exception of 'PLUS', operators only apply to numeric values. PLUS can be used to concatenate two string values.			
	MATHOP	Yes	Character	None
	Meaning: The math operation to perform on the indicated variables. If there is one VAR section, the following operations are supported: MINUS (reverses the sign of the value) If there are exactly two VAR sections, the following math operations are supported: PLUS (addition), MINUS (subtraction or sign reversal), TIMES (multiplication), DIV (division), MOD (modulus), GT (greater than), GE (greater than or equal to), LT (less than), LE (less than or equal to), EQ (equivalence), AND (and two numbers), OR (or two numbers). If there are more than two VAR sections, the following operations are supported: MIN (return the minimum of all the values), MAX (return the maximum of all the values), AVG (return the average of all the values).			

VAR				
	Meaning: Specifies a variable. It is used to identify a variable from an existing document.			
	VALTYPE	No	Character	NUMVALUE
	Meaning: The representation of the value. It can be either 'NUMVALUE' for numeric values, 'STRVALUE' for string values, or 'DOCVALUE' for document values.			
	VARNAME	Yes	Character	None
	Meaning: The name of the variable to locate.			
	OPID	No	Numeric	None
	Meaning: The ID of the operation which contains the desired variable. If this tag is not present, the OPID defaults to a predetermined value.			
	OPVARIOTYPE	No	Character	INPUT
	Meaning: The location in the document where the variable is located. The default is 'INPUT' which indicates the variable is in the input section. The other possible value is 'OUTPUT' which indicates the variable is in the output section of the document.			
	OPDOCIOTYPE	No	Character	INPUT
	Meaning: The document that contains the desired variable. This can either be the operation request document (INPUT)) or the operation response document (OUTPUT) of the indicated operation..			
	BCAST_MATHOP	No	Character	None
	Meaning:. This must be specified if the operation identified by OPID is a broadcast operation. It indicates how to treat the values held by each broadcast instance. Since only one value can be mapped to a variable, this describes the math operation to perform to determine that value. The supported operations are SUM (to add up all of the values), MIN (to return the minimum of all the values), MAX (to return the maximum of all the values), and AVG (to return the average of all the values)..			
FUNC-TION				
	Meaning: If present, indicates an external function should be executed for this expression.			
	LIBNAME	Yes	Character	None
	Meaning: The path and name of the library which contains the desired function.			
	FUNCNAME	Yes	Character	None
	Meaning: The name of the function to execute.			
	VALTYPE	No	Character	NUMVALUE
	Meaning: The type of value returned by the function. It can be either 'NUMVALUE' for numeric values, 'STRVALUE' for string values, or 'DOCVALUE' for document values.			

1   Example 15 defines an expression that always returns the value of 10.

2   Example 15:

3   <EXPR>
4   <VALUE NUMVALUE="10" />
5   </EXPR>

6   Example 16 defines an expression in which two numbers are added together from the

7   input section of the operation request document of the operation whose OPID is 1.

8   Example 16:

9   <EXPR>
10   <OPERATOR MATHOP="PLUS">
11   <VAR VALTYPE="NUMVALUE" VARNAME="VAR1" OPID="1" />
12   <VAR VALTYPE="NUMVALUE" VARNAME="VAR2" OPID="1" />
13   </OPERATOR>
14   </EXPR>

15   Example 17 defines an expression as a function called "myFunc" which is found in the

16   library at location "/usr/local/lib/myLib.so" and returns a numeric value.

17   Example 17:

18   <EXPR>
19   <FUNCTION FUNCNAME="myFunc"
20   LIBNAME="/usr/local/lib/myLib.so"
21   VALTYPE="NUMVALUE" />
22   </EXPR>

23   **4.   Operation of the Transaction Service.**

24   The general functions of transaction processing service 200 of FIG. 2 are

25   illustrated in the flowchart of FIG. 8. These functions are described below with reference

26   to the components shown in FIG. 2. CX server 10 receives messages from requesting

27   applications and service applications. CX server 10, after handling message protocol

28   functions, passes these received messages to transaction service 200 in box 202. CX

29   server 10 uses transaction service 200 to track a transaction thread, to traverse transaction

1   logic and to perform transaction execution. Transaction service 200 provides a set of

2   service interfaces with conditionals and mapping of DOM objects for working with the

3   transaction logic. The service interfaces include those shown in Table 8.

4   TABLE 8

API	Function
CreateTransaction	creates a transaction instance
StartTransaction	starts execution of a transaction instance
EndTransaction	ends execution of a transaction instance
AdvanceTransaction	advances a transaction instance by executing a next list of operations
GetTransaction	gets a transaction object
RetryTransaction	retries or re-executes selected transaction operations
RestartTransaction	restart from the beginning the selected transaction
SuspendTransaction	halt execution of an active transaction
ResumeTransaction	resume execution of an active transaction
AbortTransaction	Halt, permanently, execution of an active transaction
GetInputMsg	gets the input XML document of a transaction or operation
GetOutputMsg	gets the output XML document of a transaction or operation

5   As shown in FIG. 2, transaction service 200 handles four types of messages; it

6   may receive transaction request messages 284 and operation response messages 290, and

7   it may send operation request messages 286 and transaction response messages 294.

8   Transaction service 200 first determines what kind of message has been received

9   in the query boxes 204 and 206. If the message is neither one, control passes to a

10   message error handling procedure 250 and processing control returns to CX server 10. If

1   the message is a transaction request message 284, this is a new transaction, and control

2   passes from box 204 to box 210, where transaction service 200 calls the

3   StartTransaction interface to create a new transaction instance associated with a

4   unique identifier, referred to as the transaction ID, and to start transaction execution.

5   Creating a new transaction includes calling CreateTransaction to retrieve the

6   transaction definition 282 that matches the transaction name in request message 284 from

7   transaction definition database 296, and producing the directed acyclic graph that

8   represents the transaction instance. Transaction service 200 then begins traversing the

9   transaction instance graph by obtaining a list of operations for execution, in box 214,

10   using the SPLIT logic of the head operation. For each operation in the operation list,

11   transaction service 200 calls GetInputMSG to create the input document(s) for the

12   operation, in box 230, using the information specified in the JOIN section for the

13   operation, and produces the operation request document 286, in box 234.   For each

14   operation, transaction service determines the service application name of the service

15   application that is to perform the operation, using the CXCNAME tag. Then, transaction

16   service 200 returns the operation request document(s) CX server 10 for sending to their

17   respective service applications, in box 260, and control returns to CX server 10.

18   If the incoming message is not a transaction request, as determined in box 204,

19   transaction service 200 queries whether it is an operation response 286 received from a

20   service application, in box 206. If the message is an operation response, transaction

21   service 200 updates the transaction state. Each operation response message contains the

1   transaction ID, the operation name and the output results produced by the service

2   application, and is stored in a data store for later access and processing. The transaction

3   state changes every time an operation response message is received.

4   Transaction service 200 then determines, in box 216, whether the operation

5   response message is in response to a broadcast operation. A broadcast operation may

6   involve invoking more than one service application to perform the operation. In the

7   illustrated implementation of transaction service 200, the criteria for advancing to a next

8   operation node is to wait until all responses to operations associated with the node are

9   received. To determine whether a broadcast operation is completed, the RESPONSE,

10   MIN and EXPR tags are used to determine how to process operation response messages.

11   As noted earlier, if the RESPONSE section is present, the MIN tag specifies the minimum

12   number of responses required to be received before advancing past this operation. If no

13   RESPONSE section is defined, the default is that all operation requests sent on behalf of

14   this broadcast operation must be received before the operation as a whole can be

15   advanced. If the EXPR section is present, this indicates an expression that must be

16   evaluated against each response received to determine if it should be counted toward the

17   minimum. If no EXPR section is present, all responses received will be counted toward

18   the minimum, if a minimum has been specified. Collectively these rules may be referred

19   to as the *broadcast advance criteria*. If the query in box 216 determines that this

20   operation response message is in response to a broadcast operation, then Transaction

21   service 200, in box 218, queries whether the broadcast advance criteria have been met,

1   and if so, control proceeds to box 220 to advance the transaction. If broadcast advance

2   criteria have not been met, then this operation node is not complete, the transaction

3   cannot be advanced, and control is returned to CX server 10.

4   If the message is an operation response message and there is no pending broadcast

5   operation, transaction service 200 calls AdvanceTransaction procedure in box 220.

6   Transaction service 200 then calls GetTransaction to update the transaction

7   instance's state information, and then evaluates the SPLIT logic of the operation

8   associated with this operation response message to obtain the next list of operations from

9   the transaction instance graph. Transaction service 200 queries, in box 224, whether

10   there is a next operations list available. If a next list of operations is available, then CX

11   server 10 still has more operations to perform for this transaction, and the transaction may

12   be advanced to have the appropriate service application(s) perform the next operations.

13   Transaction service 200 ensures that none of the operations in the list of next operations

14   has a predecessor operation that is still pending and has not yet completed. Control

15   passes to boxes 230, 234 and 260 where the next operation request message(s) are

16   produced and sent out, as described above.

17   If the query in box 224 indicates that there is no next operations list, the

18   transaction has been completed (assuming that the incoming message is not in error).

19   This means that the operation response message just received is for an operation whose

20   SPLIT logic or OPLINK section points to the required tail operation in the transaction

21   instance. The tail operation node is available as the next operation in the next operation

1   list. Control then passes to boxes 240 and 244 where the transaction response message is

2   created. The JOIN logic of the tail operation node contains the transaction response

3   document format expected by the requesting (originating) application. Transaction

4   service 200 calls `GetOutputMSG` to obtain the output document for the transaction,

5   produces the transaction response message using this document format, identifies the

6   requesting application and then sends the transaction response message to that application

7   in box 260.

8   Transaction service 200 also supports transaction timer, purging, logging and

9   recovery mechanisms that are not shown in FIG. 8. The timer mechanism allows

10 transaction service 200 to discontinue processing of this transaction if operation

11 responses are not received in a timely manner. When operation responses are not

12 received after retries and a waiting period, the transaction instance is ended (i.e., removed

13 from memory) and is marked as a timeout transaction in the data store of transactions.

14 Creating a timer for each operation, if needed, is accomplished during

15 `AdvanceTransaction` procedure 220. When transactions end, there is a purging

16 mechanism to remove the transaction object from all transaction lists.

17   Transaction logging involves saving the transaction instance, sufficient run time

18 information, and transaction state changes to enable transaction recovery if a first CX

19 server 10 is no longer operational and a backup CX server assumes transaction

20 processing. The following transaction data elements are saved during transaction

21 logging: the transaction instance, the transaction request message, the transaction

1   response message (if any), and transaction attributes including the transaction ID, the

2   transaction DAG name, transaction state information, the originating requesting

3   application identifier and name. Operation information saved includes operation request

4   messages, operation response messages, the operation identifier (OPID), operation state

5   information, and broadcast operation information (e.g., number of service applications

6   requested and number of operations pending). Executing CXCs are saved as well.

7   Transaction recovery may be done at the individual transaction level or for all

8   transactions started by CX server 10.

9   **5.   Transaction tools.**

10   **a.   Test transaction template.**

11   Table 9 provides a transaction that may be used as a template to test the interface

12   between a service application (referred to as a CXC) and transaction service 200 of CX

13   server 10. Transaction definition line `<CXTXDAG NAME="TsGraph"` identifies this

14   XML document as containing a transaction definition. Transaction definition line

15   `<TRANSACTION NAME="TX-Template">` identifies the start of a transaction

16   definition section for a transaction called "TX-Template". TX-Template is the name to

17   be used in the transaction request document to request a transaction of this type to be

18   executed. The template transaction is a simple linear transaction with a total of three

19   operations identified by the `OPID` tag as operations 0, 1 and 2. The operations identified

20   with `OPID = 0` and `OPID = 2` and having names `"CxtsHeadOp"` and

21   `"CxtsTailOp"` indicate the head and tail operations, respectively, required in all

1   transactions.   The   transaction   definition   line   `<OPLINK    SRCOPID="0"`

2   `DSTOPID="1"` specifies that the operation having `OPID = 1` is the destination

3   operation of the head operation, and so is the first "real" operation to be performed.

4   Thus, whenever a transaction request document is received requesting a transaction called

5   "TX-Template", the first operation to be executed is `OPID = 1`, which requests the

6   CXC   being   tested   to   perform   its   function.   The   transaction   definition   line

7   `</OPERATION>` indicates the end of an operation definition.

8   TABLE 9

```
<CXTXDAG NAME="TsGraph"
<TRANSACTION NAME="TX-Template"

<OPERATION OPID="0" NAME="CxtsHeadOp" CXCNAME="Head">
<OPLINK SRCOPID="0"   DSTOPID="1"   /
</OPERATION>

<!-- The operation below should be customized to the
CXC being tested. -- >
<!-- The following must be specified in the OPERATION
section to complete this transaction definition: -->
<!-- * NAME - name of the operation.  Typically, this
is the same as the name of your CXC; and -->
<!-- * CXCNAME of the CXC.  This is the name of the
CXC being tested. -->
<OPERATION OPID="1" NAME="cxc-template-op-1"
CXCNAME="*">
<OPLINK SRCOPID="1" DSTOPID="2" />
<JOIN DOCVARTYPE="VARLIST> </JOIN>
</OPERATION>

<OPERATION OPID="2" NAME="CXtsTailOp" CXCNAME="">
<JOIN DOCVARTYPE="VARLIST"></JOIN>
</OPERATION>
</TRANSACTION>
</CXTXDAG>
```

9   The operation definition for `OPID = 1` further specifies the name of the

10   operation, `"cxc-template-op-1"`, and the name of the CXC designated to perform

1　the operation. Typically, the NAME and CXCNAME fields will have the same value.

2　So, for example, if the CXC to be executed is named 'Parse', the line would read:

3　　　　　`<OPERATION OPID="1" NAME="Parse" CXCNAME="Parse">`

4　The JOIN tag identifies how this `<JOIN>` operation should build its operation

5　request document. In this case, this type of join will map all of the variables in the data

6　section of the transaction request document into the data section of the operation request

7　document. For example, assume the transaction request document has the following data

8　section:

```
9       <DATA>
10      <INPUT>
11      <FILENAME NAME="myTestFile.txt" />
12      <MAPNAME NAME="myMap.txt" />
13      </INPUT>
14      <OUTPUT> </OUTPUT>
15      </DATA>
16
```

17　Then, the data section of the operation request document would look like:

```
18      <DATA>
19      <INPUT>
20      <FILENAME NAME="myTestFile.txt" />
21      <MAPNAME NAME="myMap.txt" />
22      </INPUT>
23      <OUTPUT> </OUTPUT>
24      </DATA>
```

25　The operation request document for operation 1 is then sent to the CXC being

26　tested. The transaction definition line `<OPLINK  SRCOPID="1"  DSTOPID="2"`

27　specifies that the operation having OPID = 2 is the destination operation of operation 1,

28　and so is to be performed after completion of operation 1.

29　A service application has access to a library of interfaces, referred to as the

30　CXsdk, to interact with CX server 10. The interfaces enable a custom-built application to

1   include the application logic for the function of the CXC. The interfaces also provide for

2   a service application to connect to CX server 10, to register itself as a CXC to sign up for

3   an operation, and to send and receive messages. The CXsdk library also includes the

4   XML parsing and constructing tools. The service CXC uses one of these interfaces to

5   extract any needed variables from the operation request document it receives from CX

6   server 10. The CXC then performs the function it has been configured to do. After

7   completing its function, the CXC issues an operation response document, using the

8   CXsdk, and sends that document back to CX server 10, and to operation 2, the tail

9   operation, in the transaction graph of the TX-Template transaction.

10   The role of the tail operation is to build the transaction response document that is

11   sent back to the CXC that requested the transaction providing information about the

12   transaction that just executed. The join section for operation 2 identifies how to build the

13   transaction response document. In this case, this type of join will map all of the variables

14   in the data section of the operation response document from operation OPID = 1. CX

15   server 10 then sends the transaction response document back to the requesting CXC.

16   Finally, the transaction definition lines </TRANSACTION> and </CXTXDAG>

17   respectively identify the end of the transaction definition for TX-Template, and the end of

18   the XML document.

19   **b.   Producing transaction definitions.**

20   A user interface application may be provided with CX server 10 for a user to

21   design and specify a transaction definition. A user may construct a transaction visually

1   by selecting from among operations that are available, according to their descriptions.

2   The user interface may provide assistance to the user when specifying a unique identifier

3   for the transaction definition. In addition, the user must identify the input and output

4   DTD's of the transaction. The user interface may operate in one of two modes. The first

5   is simply as a data entry tool for capturing the transaction definition. The second mode of

6   operation allows for the user interface to validate the correctness of the definition

7   dynamically, by connecting to the CX server.

8   **6.   The machine and software product of the invention.**

9   FIG. 9 is a block diagram of distributed network 140 that includes processor-

10   controlled machines 142, 144, 146 and 100. The component parts of machine 100 have

11   been enlarged to schematically illustrate a machine in which the present invention may be

12   used. Machine 100 is an example of a processor-controlled machine that may be used to

13   implement commerce exchange server 10 of FIG. 1 including transaction service 200

14   which utilizes the transaction DAG data structure of the present invention. Similarly, any

15   one of the processor-controlled machines 142, 144 and 146 may implement one of

16   machines 20, 22 or 24 of FIG. 1 that include a service application or one of machines 32

17   or 36 that include a client application of the commerce network illustrated in FIG. 1.

18   While the present invention may be used in any machine having the common

19   components, characteristics, and configuration of machine 100, the invention is not

20   inherently related to any particular processor, machine, system or other apparatus. The

21   machine or system may be specially constructed and optimized for the purpose of

22   carrying out the invention. Alternatively, machine 100 may comprise a general-purpose

1   computer selectively activated or reconfigured by a computer program stored in the

2   computer. In still another alternative machine 100 may be a combination of a general-

3   purpose computer and auxiliary special purpose hardware. When a machine such as

4   machine 100 is suitably programmed to embody or make use of the present invention, the

5   machine is not a standard or known configuration. In the claims, machine 100 is referred

6   to as a "computer" for purposes of simplifying the claim language, but the term

7   "computer" is intended to include any and all machines as described and shown in FIG. 9

8   and is not intended to limit the scope of machine 100 in any way.

9   Machine 100 includes a bus or other internal communication means 101 for

10   communicating information, and a processor 102 coupled to bus 101 for processing

11   information. Machine 100 further comprises a random access memory (RAM) or other

12   volatile storage device 104 (referred to as main memory), coupled to bus 101 for storing

13   information and instructions to be executed by processor 102. Main memory 104 also

14   may be used for storing temporary variables or other intermediate information during

15   execution of instructions by processor 102. Machine 100 also comprises a read only

16   memory (ROM) and/or static storage device 106 coupled to bus 101 for storing static

17   information and instructions for processor 102, and a data mass storage access device 107

18   such as a magnetic disk drive or optical disk drive. Data mass storage access device 107

19   is coupled to bus 101 and is typically used with a computer readable mass storage

20   medium 160, such as a magnetic or optical disk, for storage of information and

21   instructions. Machine 100 may include more than one storage access device 107. For

22   example, machine 100 may include both a storage access device for a non-removable

1   medium such as an internal magnetic (hard) disk and a mass storage access device for a

2   removable medium such as an optical CD-ROM, a magnetic floppy disk, a PC-card, or

3   magnetic tape.

4   Machine 100 may, but need not, include a conventional display device 121

5   capable of presenting images, such as a cathode ray tube or a liquid crystal display (LCD)

6   device or any other device suitable for presenting images. Display device 121 is coupled

7   to bus 101 through bus 103 for displaying information to a computer user. An

8   alphanumeric input device 122, including alphanumeric and other keys, may also be

9   coupled to bus 101 through bus 103 for communicating information and command

10   selections to processor 102. An additional user input device is cursor control device 123,

11   such as a mouse, a trackball, stylus, electronic tablet, or cursor direction keys coupled to

12   bus 101 through bus 103 for communicating direction information and command

13   selections to processor 102, and for controlling cursor movement on display device 121.

14   Another device which may optionally be coupled to bus 101 through bus 103 is a hard

15   copy device 124 which may be used for printing instructions, data, or other information

16   on a medium such as paper, film, or similar types of media. Note that the actual manner

17   in which the physical components of machine 100 are connected may vary from that

18   shown in FIG. 9. The manner of connection may include hardwired physical connections

19   between some or all of the components, as well as connections over wired or wireless

20   communications facilities, such as through remote or local communications networks and

21   infrared and radio connections. Note further that not all of the components of machine

22   100 shown in FIG. 9 may be required to carry out the functions of commerce exchange

1   server 10 or to make use of the transaction DAG data structure of the present invention.

2   Those of ordinary skill in the art will appreciate that various configurations of machine

3   100 may be used to carry out a particular implementation of commerce exchange server

4   10 or the transaction DAG data structure. For example, machine 100 may be a

5   Workgroup Enterprise server machine manufactured by Sun Microsystems, Inc. of

6   Mountain View California that includes one or more Ultra SPARC™ processors, and that

7   operates using the Solaris™ operating system.

8   Machine 100 further includes communication, or network interface, device 125,

9   coupled to bus 101 through bus 103, for use in sending data to and receiving data from

10 other nodes of distributed network system 140 according to standard network protocols.

11 This communication device 125 may include any of a number of commercially available

12 networking peripheral devices such as those used for coupling to an Ethernet, token ring,

13 Internet, or wide area network.

14   Processor 102, together with an operating system, operates to execute instructions

15 (e.g., program code) to produce and use data. The program code and data may reside in

16 main memory (RAM) 104, in read only memory 106, on the non-removable hard disk

17 storage accessed by storage access device 107, or even on another processor-controlled

18 machine connected to network 140. The program code and data may also reside on a

19 removable medium that is loaded or installed onto machine 100 when needed by means

20 of a storage access device 107 suitable for that purpose. When program code (i.e.,

21 software) implementing commerce exchange server 10 is stored in a memory device

22 accessible to processor 102, machine 100 is configured to perform the functions of

1   commerce exchange server 10 of FIG. 1, and, in particular, to process transactions having

2   the structured format illustrated in FIG. 5 or FIG. 6 and FIG. 7. An input transaction

3   request message, such as transaction request message 284 of FIG. 2, is provided from

4   communication device 125 and is forwarded via data bus 103 to bus 101 for storage in

5   main memory 104 for later access by processor 102. Processor 102 executes program

6   instructions, included in one of the above-described memory components, that implement

7   operation 200 of FIG. 8. During execution of the instructions, processor 102 accesses

8   memory 104 or 106 to obtain or store data necessary for performing its operations. For

9   example, when machine 100 is configured to perform operation 500 of FIG. 8, processor

10   102 may access transaction DTD 50 (FIG. 5) or transaction DTD 80 (FIG. 6 and FIG. 7)

11   in memory 104 in order to perform the functions of transaction service 200 starting a new

12   transaction instance.

13   FIG. 9 also shows software and data structure product 160, an article of

14   manufacture that can be used in a machine that includes components like those shown in

15   machine 100. Software and data structure product 160 includes data storage medium 170

16   which stores instructions, also referred to as program code or computer readable code, for

17   executing operations that process transactions as defined by the present invention, such as

18   operation 200 of FIG. 8. Data storage medium 170 also stores one or more data

19   structures, such as Transaction DAG 50 of FIG. 5 for use in producing transaction

20   instances and executing transactions. As used herein, a "data storage medium" covers

21   one or more distinct units of a medium that together store a body of data. Examples of

22   data storage media include magnetic media such as floppy disks, diskettes, magnetic tape,

1   and PC cards (also previously known as PCMCIA memory cards), optical media such as

2   CD-ROMs, and semiconductor media such as semiconductor ROMs and RAMs. By way

3   of example, a set of magnetic disks or optical CD-ROMs storing a single body of data

4   would be a data storage medium.

5   Software and data structure product 160 may be commercially available to a

6   purchaser or user in several forms. In one typical form, software and data structure

7   product 160 is commercially available in the form of a shrink-wrap package that includes

8   data storage medium 170 and appropriate documentation describing the product. In that

9   case, data storage medium 170, also referred to as a computer-readable medium, is a

10   physical medium that stores one or more data structures or instruction data that is

11   accessed by storage medium access device 107 or its equivalent. "Storage medium access

12   device" is a device that can access data stored on a data storage medium. Storage

13   medium access device 107 may be contained in a distinct physical device into which data

14   storage medium 170 is inserted into, mounted on, or otherwise installed into, in order for

15   the storage medium access device to access and retrieve the data stored thereon.

16   Examples of storage medium access devices include disk drives, CD-ROM readers, and

17   DVD devices. A storage medium access device may be physically separate from

18   machine 100, or enclosed as part of a housing of machine 100 that includes other

19   components. Mass storage device 107 may also be remotely located (not shown) as part

20   of some other processor-controlled machine, such as a server, on network 140. Mass

21   storage device 107 may provide instructions retrieved from medium 170 to processor 102

22   via bus 101, causing processor 102, when executing the instructions, to process

1   transactions in accordance with the teachings herein. Mass storage device 107 may

2   provide one or more data structures retrieved from medium 170 to processor 102 via bus

3   101, for use in processing transactions in accordance with the teachings herein. If device

4   107 is remotely located, program instructions and data structures are provided from

5   storage medium 170 to processor 102 of machine 100 by way of communication device

6   125 from network 140.

7   Software and data structure product 160 may also be commercially or otherwise

8   available to a user in the form of a data stream indicating instruction data for processing

9   transactions or one or more data structures for use in processing transactions in

10   accordance with the teachings herein. The data stream is transmitted to the user over a

11   communications facility from a remotely-located storage device. In this case, article 160

12   is embodied in physical form as signals stored on the remotely-located storage device; the

13   user accesses the contents of data storage medium 170 in order to purchase or otherwise

14   obtain a copy of those contents, but typically does not purchase or acquire any rights in

15   the actual remotely-located storage device. When software product 160 is provided in the

16   form of a data stream transmitted to the user over a communications facility from the

17   remotely-located storage device, instruction data and data structures stored on data

18   storage medium 170 are accessible via communications device 125. Alternatively, a data

19   stream transmitted to the user over a communications facility from the remotely-located

20   storage device may be stored in some suitable local memory device of machine 100 or a

21   data storage medium 107 locally accessible to processor 102 using bus 101.

1   FIG. 9 illustrates various examples of how data storage medium 170 may be

2   configured. Software and data structure product 160 may include one or more of the

3   types of data illustrated in FIG. 9. For example, data storage medium 170 may be

4   configured with transaction definition data structure 280 of FIG. 2 and transaction DTD

5   data structure 50 of FIG. 5 for use by transaction service 200 for producing a transaction

6   instance DAG data structure and executing a transaction according to the process flow in

7   FIG. 8. When implementing the specific illustrated embodiment of CX server 10 and

8   transaction service 200 described herein, data storage medium 170 would be configured

9   with transaction DAG data structure 80 of FIGS. 6 and 7.

10   Data storage medium 170 may also be configured with transaction service

11   processing instruction data 162 for performing operation 200 (FIG. 8). FIG. 9 shows

12   representative examples of the functional components of instruction data 162 such as

13   advance transaction instructions 164 and start transaction instructions 166. The

14   instruction data 162, 164 and 166 is provided to processor 102 for execution when

15   transaction service processing is to be performed. For example, when instructions 162

16   are provided to processor 102, and processor 102 executes them, machine 100 is operated

17   to perform the operations for starting or advancing a transaction, processing a broadcast

18   transaction, producing operation request messages, or producing a transaction response

19   message, according to operation 200 of FIG. 8. Note also that when software and data

20   structure product 160 comprises the entire commerce exchange server application 10 of

21   FIG. 1, data storage medium 170 may include additional instruction data (not shown) for

22   carrying out operations 12, 14, 19 and 400 of CX server 10.

1   While the invention has been described in conjunction with one or more specific

2   embodiments, this description is not intended to limit the invention in any way.

3   Accordingly, the invention as described herein is intended to embrace all modifications

4   and variations that are apparent to those skilled in the art and that fall within the scope of

5   the appended claims.